CAMPUS
DE EXCELENCIA
INTERNACIONAL

Master Universitario en Software y Sistemas

Universidad Politécnica de Madrid

Facultad de Informática

TRABAJO FIN DE MASTER

Evaluation of mission planning strategies of a robotic aerial vehicle for the IARC international competition

Author:
Carlos Villalba Coronado

Director:
Martín Molina González

MADRID, MAY 2015

INDEX

1. INTRODUCTION....1
1.1. SCOPE OF THE WORK....2
1.2. Document Structure....3
2. PURPOSE....4
3. STATE OF THE ART....5
3.1. Robotics Overview....5
3.2. Robotic Paradigms....7
3.2.1. Hierarchical Paradigm....7
3.2.2. Reactive Paradigm....8
3.2.3. Hybrid Paradigm....9
3.3. Autonomous Unmanned Aerial Vehicles....10
3.3.1. Autonomy Levels....12
3.3.2. Autonomy Levels For Unmanned Rotorcraft Systems (ALFURS)....14
3.3.3. Research Groups....18
3.4. IARC - International Aerial Robotics Competition [8]....21
3.4.1. First Mission (1990 – 1995)....21
3.4.2. Second Mission (1996 – 1997)....22
3.4.3. Third Mission (1998 – 2000)....22
3.4.4. Fourth Mission (2001 – 2008)....23
3.4.5. Fifth Mission (2009)....24
3.4.6. Sixth Mission (2010 -2013)....24
3.5. Seventh Mission Description [9] [10]....25
4. SIMULATION PLATFORM [10]....29
4.1.1. Architecture Design....29
4.1.2. ROS – Robot Operating System [11]....31
4.1.3. Simulator Implementation....36
5. IMPROVING THE SIMULATION PLATFORM....51
5.1. Time....51
5.2. Network optimization....57
5.3. Speed-Up Real-Time Simulation System....58

5.4. Evaluation 58
6. STRATEGIES DEVELOPMENT 60
6.1. Mission Time 60
6.2. Searching Strategies 61
6.2.1. Center Point Strategy 61
6.2.2. Random Strategy 61
6.2.3. Non Visited Strategy 61
6.2.4. Square Route Strategy 62
6.2.5. Diamond Strategy 62
6.2.6. More Robots Strategy 63
6.2.7. Last Robot Seen Strategy 63
6.2.8. First Near Strategy 64
6.2.9. First Far Strategy 65
6.2.10. Route Strategy Alternative 65
6.2.11. Combined Strategies 66
6.3. Selection Strategies 66
6.3.1. First Near Choice 66
6.3.2. First Far Choice 67
6.3.3. Discarding Guided Robots 67
6.3.4. Discard Near Obstacles 68
7. EVALUATION 69
7.1. Evaluation Method 69
7.2. Time Influence 70
7.3. Selection Strategies 71
7.4. Searching Strategies 72
7.4.1. Center Point 74
7.4.2. Random and Non Visited Strategies 75
7.4.3. Route Strategies 75
7.4.4. First Far and First Near 75
7.4.5. Combined Strategies 76
7.4.6. Distribution of Data 77

8. CONCLUSIONS AND FUTURE WORK 79
9. Bibliography.....83
10. LIST OF ACRONYMS.....85

ABSTRACT

The International Aerial Robotics Competition (IARC) is an important event where teams from universities design flying autonomous vehicles to overcome the last challenges in the field. The goal of the Seventh Mission proposed by the IARC is to guide several mobile ground robots to a target area. The scenario is complex and not determinist due to the random behavior of the ground robots movement. The UAV must select efficient strategies to complete the mission.

The goal of this work has been evaluating different alternative mission planning strategies of a UAV for this competition. The Mission Planner component is in charge of taking the UAV decisions. Different strategies have been developed and evaluated for the component, achieving a better performance Mission Planner and valuable knowledge about the mission.

For this purpose, it was necessary to develop a simulator to evaluate the different strategies. The simulator was built as an improvement of an existing previous version.

SUMMARY IN SPANISH

La Universidad Politécnica de Madrid está participando en la Misión 7 de la International Aerial Robotics Competition. Este Trabajo Fin de Máster está centrado en la mejora del planificador de misiones que opera en el vehículo robótico aéreo, encargado de tomar decisiones de forma autónoma. Concretamente ha consistido en el estudio de diferentes estrategias que el robot participante realiza durante la competición. Los resultados obtenidos han dado como resultado un planificador con un rendimiento general alto, además de la adquisición de conocimiento valioso que podrá ser utilizado para futuras estrategias más eficientes. Además del estudio de estrategias, este Trabajo Fin de Master también incluye la mejora del simulador desarrollado por la UPM, habilitando una opción para que se ejecute de forma acelerada.

La competición en la que se participa es la International Aerial Robotics Competition, una prestigiosa competición internacional de vehículos aéreos autónomos. Esta competición se celebra anualmente y se basa en misiones. La misión actual, en la que participa el equipo de la UPM, es la número siete. Esta misión consiste en que el vehículo aéreo participante debe guiar de forma completamente autónoma a una serie de robots terrestres hacia un área objetivo. Estos robots terrestres están en continuo movimiento de forma preestablecida y parcialmente aleatoria. La forma que el vehículo aéreo tiene que guiarlos es mediante determinadas interacciones físicas con ellos que modifican su movimiento. Además de esto, el escenario tiene varios robots que actúan como obstáculos móviles que el vehículo aéreo debe evitar para no ser descalificado. El vehículo aéreo tendrá que guiar el máximo número de robots terrestres hacia el área objetivo para ganar la competición.

Para el diseño del Mission Planner se han propuesto e implementado distintas estrategias que el vehículo aéreo puede utilizar. En primer lugar, se han propuesto estrategias de búsqueda de robots candidatos para guiarlos al objetivo. El planificador se encarga de intentar ir al mejor sitio posible con la información disponible. Las estrategias propuestas utilizan distintos criterios. Por ejemplo, la utilización de rutas que el vehículo recorre buscando los objetivos terrestres, la priorización de unas zonas sobre otras o la utilización de la última posición conocida de un robot para dirigirse al área dónde se encontraba. En complemento con estas estrategias de búsqueda, se han realizado estrategias de selección entre varios robots visibles, así como criterios para dejar de guiarlos en determinadas circunstancias.

Estas estrategias han sido evaluadas para comprobar su eficacia. La evaluación ha consistido en la realización de una gran cantidad de ejecuciones con las distintas configuraciones y el estudio posterior de los resultados obtenidos. Los resultados se miden en función del número de robots guiados y de algunos datos auxiliares, como el número de robots que han salido del escenario o el tiempo necesitado para guiar los

siete robots mínimos para completar la misión. Además, es necesario realizar una gran cantidad de ejecuciones con cada configuración para conseguir datos suficientemente fiables sobre el comportamiento de cada estrategia, debido a que el escenario es no determinista y por tanto único en cada simulación.

Para validar el Mission Planner, la UPM disponía de un simulador desarrollado en el propio grupo de investigación. Este simulador de tiempo real se utiliza para poder crear un escenario virtual que se comporte de forma similar a la competición. Gracias a este simulador se pueden desarrollar algunos componentes software, como el Mission Planner, sin las importantes dependencias de los elementos hardware del robot como sensores y actuadores, así como los otros elementos físicos de la competición. Para realizar este simulador, se ha utilizado un popular middleware especializado en robótica llamado ROS (Robot Operating System). Como el UAV también utiliza este middleware, la integración de los componentes desarrollados a partir del simulador es sencilla.

El mayor problema que presentaba el simulador previamente desarrollado por el propio grupo de investigación en la UPM era la gran cantidad de tiempo que necesitaba para realizar cada simulación. Al ser un sistema de tiempo real, su tiempo de ejecución dependía directamente del reloj de la misión, llegando a superar los diez minutos máximos que puede durar la misión. Para evaluar las distintas estrategias diseñadas era prioritario disminuir el tiempo necesario de simulación con el fin de poder realizar una gran cantidad de simulaciones. Para conseguir este propósito, se realizaron varios cambios en el simulador. El objetivo era conseguir que actuase de la misma forma que el original, pero aprovechando todo el tiempo de computación. Para ello se ha eliminado el sistema de tiempo real que esperaba a que pasara un determinado tiempo para empezar un nuevo ciclo. Utilizado un reloj que avanza de forma simulada y realizando varios cambios en la arquitectura de mensajes y los sistemas de sincronización, se ha conseguido un simulador que funciona de forma similar, pero que optimiza el procesador. En nuestra máquina, un ordenador con un Intel Core i7-3610QM, conseguimos ejecutarlo 25 veces más rápido que su versión anterior.

Los resultados de la evaluación nos han permitido encontrar algunas soluciones de alto rendimiento. Concretamente se ha encontrado una configuración que ha guiado al menos siete robots un 93% de las veces, con una media de 7,9 robots. Sin olvidar que se ha obtenido de los experimentos un conocimiento muy valioso para proponer nuevas estrategias. Desgraciadamente el tiempo es limitado y el estudio se ha realizado sobre el rendimiento general del planificador. Por tanto, queda pendiente realizar estudios más profundos sobre aspectos concretos de las estrategias. Del mismo modo que seguir desarrollando y evaluando estrategias, para mejorar lo más posible el planificador.

1. INTRODUCTION

In the last years, the importance of autonomous vehicles as research field is increasing. Thanks to the advances in different areas like materials, computer vision or communications, autonomous vehicles are getting more and more progress, becoming more useful for real applications. In the case of aerial autonomous vehicles, their flight capacity gives advantages of mobility and privileged vision from the altitude. They can perform many civil applications like aerial mapping, traffic surveillance or cinematography, among others. Even the achievements of autonomous vehicles are notorious, still are far to replace humans for general purpose situations which require complex adaptability to unexpected conditions.

Since 1991, the International Aerial Robotics Competition (IARC) has been celebrated to boost the research in autonomous aerial vehicles. Thanks to the competition, most important universities all over the world have focused on overcoming the different challenges proposed by the competition organization. In 2013, the IARC proposed the seventh mission, last mission until now. For accomplish the previous mission, the robots completed precision tasks. Those tasks required lot of precision and control. The new mission goes further, and proposes complex interactions in a dynamic scenario. The vehicles designed by the participants must guide several ground robots with a partially random behavior to a goal area in order to complete the mission. In addition, time limitation and moving obstacles makes the challenge even harder.

The Universidad Politécnica de Madrid (UPM) has created a team which is participating in this competition. There are several challenges to overcome for completing the IARC mission. One of the most important challenges is related with computer vision. Identify moving targets and predict their future positions require complex algorithms. Other obstacle is flight control under a dynamic scenario which is always changing. The vehicle must interact with ground robots which are always moving and changing their direction. Moreover, the vehicle must avoid touching moving obstacles. The flight control needs to be accurate, fast and able to adapt to the changes in the scenario. This mission also includes as novelty an important strategy element.

The competition goal is to guide the maximum number of ground robots to the designated target area. Maximize this goal requires that the aerial vehicle takes autonomously several efficient decisions. It has to take decisions like which ground robots have higher priority to being guided or how to explore the arena if there is not any robot in the vision range. Generalizing, the vehicle has to decide the best action to perform next. To take this kind of decision, it uses the current information of the scenario from the sensors, but it can also use relevant previous knowledge like where

the ground robots have been seen or which areas have been visited. The software component in charge of taking these decisions is the Mission Planner.

The Master Degree Project described in this document is focused in the research and development of the mission planner for the Seventh Mission of the IARC. This project is part of the graduation work of a Master Degree Program of the Universidad Politécnica de Madrid called Master Universitario en Software y Sistemas. Following, the scope of the work and the structure of the document are described in more detail.

1.1. SCOPE OF THE WORK

The work realized for this Master Degree Project includes several goals, described as follows:

1. **Review.** Previously to development stages, a review work has been realized. On one hand, the review phase helps to understand the current status of the field. The knowledge acquired in the review phase is critical to work in the specific field of aerial robotics. Thanks to the knowledge collected by researchers is possible to face the different issues presented with a better understanding. On the other hand, review phase also includes adaptation to the environment. This Master Degree Project continues with improvements to the simulator previously designed by the UPM team (see Section 4) and its Mission Planner component. To know deeply the previous work is needed for the project as well as learn how to use tools and programming languages, specially the Robot Operating System (see Section 4.1.2).
2. **Building a Simulation Platform**. To develop and experiment satisfactorily with the Mission Planner component, an adequate Simulation Platform is required. The Simulator developed by the UPM team had great properties for our purposes. However, it works like a real time simulator. This means that executing a single simulation needs ten minutes in most of the cases. Shortening this simulation time was highly desired. To adapt better the simulator to our requirements, changes in the architecture and other optimizations have been done as part of this Master Degree Project. The new version designed can be executed in a little portion of the time; it depends on the machine computation power.
3. **Design and implement Mission Planner Strategies.** Mission Planner is the component in charge of taking the different decisions during the competition. It works autonomously as an intelligent system. Mission Planner can use the information taken by the sensors of the robot as well as the experience acquired to plan what to do next. There are different decisions taken by the Mission Planner component. Each of these decisions can use different strategies. As part

of this work, different strategies have been proposed and implemented. The goal is to obtain useful knowledge and a higher performance Mission Planner for the competition. To achieve it, different strategies must be evaluated.

4. **Evaluate the Strategies performance.** Once the different strategies are prepared, they have been evaluated. This requires a large number of executions for comparing the different strategies behavior. The results provides metrics to quantify the performance of each strategy. This phase extracts useful knowledge about the mission. The knowledge is also used to design new strategies resulting in an iterative process.

1.2. Document Structure

This document is structured in different sections for splitting adequately the contents of the work done. This first section is the introduction. The Section 2 summarizes the general purpose of the project. Section 3 describes the State of the Art. It collects relevant research works about the topics involved in the project. Specifically, it provides a general overview of the robotics field and more detail information about autonomous robots, aerial robots and programming paradigms for robotics. Additionally, it reviews achievements and history of the International Aerial Robotics Competition, as well as describes in detail the rules of the proposed mission. The Section 4 explains the Simulation Platform previously developed by the UPM team. It includes the architecture of the platform, the main features of the different tools used and the detailed description of the different components. Special focus is done on the Mission Planner component as main target of this work. It is described in Section 4.1.3.3.

The sections after the description of the Simulation Platform, describe the work realized within the scope of this project. The Section 5 is devoted to the improvements in the Simulation Platform. It explains the different changes to achieve an improved version of the Simulation Platform according to our requirements. Specifically, the changes were realized in order to obtain a simulator with faster execution benefiting from the computer power. Section 6 includes the different strategies proposed for the mission and Section 7 the evaluation of these strategies with their results and analysis after the experiments. Finally, in Section 8 the conclusions of the project and the possible lines to follow in future work are explained. The bibliographic references used are listed in the last section (Section 9).

2. PURPOSE

Develop fully autonomous vehicles which are able to complete complex tasks is one of the main goals of robotics. However, it is a huge and complex duty with many obstacles to overcome. It needs decades from the starting investments in research to see this type of vehicles available to the users. One of the ways to boost the research in this area is the competition. In the case of aerial vehicles, the International Aerial Robotics Competition (IARC) has been guided the research of aerial robots for more than twenty years.

This Master Degree Project is part of a bigger project that involves a team from the Universidad Politécnica de Madrid (UPM) in order to compete in the International Aerial Competition. The goal of the current 7^{th} Mission of the IARC is to advance in topics like vision, interaction or planning under complex scenarios. The scenario of this tournament requires advanced interactions with many agents moving in partially random manner. Then, the interaction with these agents includes important issues difficult to predict and adapt.

As part of the project, this work is focused on develop efficient planning strategies for controlling the aerial vehicle under the competition scenario. Beyond the particular strategies obtained to solve this problem, more general knowledge can be inferred from this work. Specifically, different experiments have been executed under a simulator in order to obtain valuable knowledge to develop more refined strategies.

3. STATE OF THE ART

This section contains relevant information about the current research status of the field and some previous work which has been taken as a basis for the Master Degree Project. A general overview of robotics is briefly explained in Section 3.1. Next to the overview, in Section 3.2, robotics paradigms are described in order to understand the particularities of robot developing. From these general robotics ideas, Section 3.3 goes deeper to the particularities of Autonomous Unmanned Aerial Vehicles. The Section 3.4 outlines about the International Aerial Robotics Competition (IARC) with emphasis in the achievements and challenges overcame by the participants across its history. At last, the Section 3.5 describes the rules of the current 7th Mission in detail.

3.1. Robotics Overview

Robotics is a field of mechanical engineering, electrical engineering and computer science addressed to developing robots. Robots are devices which receive input from sensors and influence the environment around them thanks to its actuators. Robotics is a significant research field which involves many different areas such as Computer Vision, Artificial Intelligence, Locomotion, Manipulation or Sensor Development among many others. There are several features and classifications for robots. Following is a brief description of some of the uses and features of robots, including the roles they are taking in the society.

Despite robots are older, they became popular since the 1970's with the Industrial Robotics. Industrial Robotics develops machines which are able to make mechanical and repetitive tasks efficiently. Thanks to these robots the costs of many products have decreased significantly. Nowadays robots are increasing their capabilities and are able to realize more complex tasks. Therefore, new uses have appeared in the robotic field. Many of these new applications are part of the Service Robotics which develops robots with the capacity to provide different services. In this category, autonomous vacuum cleaners are becoming popular, but there are also many other applications as autonomous cars which is now an important research field.

Today, robots are being developed for many different applications. For instance, in healthcare area, disability robots can help elderly or disabled people to become more autonomous. Also, there are starting to appear robots to guide people through a museum or similar places providing interactive and personalized information. In other fields like packaging, cleaning or agriculture, robots are being developing in order to automate these tasks partially or completely which would suppose a significant impact in the socioeconomic model for at least these areas. In military applications, robots are taking more and more roles, since the bomb disposal robots which can deactivate bombs

without life risk, to many types of Unmanned Aerial Vehicles (UAV), military planes which normally are controlled remotely.

Particularly interesting are those called as Intelligent Robots. These kinds of robots work as intelligent agents. An agent is an autonomous entity which can sense its environment and acts upon it to achieve the designated goals. An intelligent agent is an agent who in addition has the ability to learn or use knowledge to achieve their goals. Robots are normally autonomous agents which use their sensors to perceive from the environment and actuators to interact with it.

Another field that deserves a mention is Humanoid Robotics. Humanoid robots are created by human inspiration. This field tries to recreate some aspects of humans in robots. The main advantages of this field are two. First, there is a psychological aspect; human-like robots are not seen as simple machines. These robots could be better accepted to interact with humans. The other big advantage for humanoid robots is that the world is designed for humans. Tasks like going upstairs, opening a door or writing with a pencil are easy for humans, but represent a challenge for robots. These tasks are possible due to how the human body is. Build a robot similarly to humans simplifies its operation through the world doing human tasks without adapting the world for them.

As most of general tasks requires move from one point to another, mobility is an important feature for robots. Mobile robots are those robots which can move by themselves. There a huge number of different possibilities for displacements. There are wheeled and tracked robots. There are also legged robots. Legged robots normally come from animal inspiration. There are, since only one leg robots which need to keep jumping to maintain the stability, to many legs robots similar to spiders or insects. Even, there are robots moving without legs similar to snakes. In addition to the land robots, there are swimming robots, aerial robots and even space robots.

Another important concept to define the capabilities and complexity of a robot are the degrees of freedom. In three-dimensional space, a free rigid object has six degrees of freedom. It can move along the three axes and also rotate over each one, which gives a total of six degrees of freedom. However, many times there are restrictions. For instance, a car can only move along one axis going forward and back. And it can rotate over the vertical axis, rotating the direction. Then, a car has two degrees of freedom, still enough to move through two-dimensional space. The number of degrees of freedom of a robot is the sum of total degrees of freedom of its parts. Many robots, like arms robots, have different components connected between them.

This Master Degree Project is based on a fully autonomous intelligent aerial robot. Autonomy degrees are explained in the Section 3.3 as well as other issues related with aerial autonomous robots. In addition, to better understand the evolution of aerial

robots, in the Section 3.4 the International Aerial Robotics Competition history is described with emphasis on the research achievements during the last 20 years. In the following pages the different robotics paradigms are explained in order to understand robot architecture design choices.

3.2. Robotic Paradigms

Murphy [1] defined a paradigm as "*a philosophy or set of assumptions and/or techniques which characterize an approach to a class of problems*". Thus, none paradigm is completely correct, but can fit better or worse with each specific problem. Identify and apply the paradigm which fits best is a central point to solve problems. For this reason, it is important to know and understand the different alternatives used in the field of robotics.

In robotics, there are three different paradigms: Hierarchical or Deliberative Paradigm, Reactive Paradigm and Hybrid Paradigm. These three paradigms are defined according to how the sensor data is perceived and distributed through the system. In robotics field, is commonly accepted the division of the robots functionality into three general categories: Sense, Plan and Act. Sense is in charge of taking the sensor data from the sensors and transforming them into useful information for the other functions. Act is in charge of robot actuators. And Plan uses the information available to produce tasks to be done by the robot. Robotics paradigms can be defined using the interaction between these three primitives.

3.2.1. Hierarchical Paradigm

Hierarchical or Deliberative Paradigm is the first paradigm used for intelligent robots. Between 1967 and 1990 it was the most used. It is inspired in the introspective thinking of the people. Under this paradigm, the robot starts sensing the world. Then, it plans the next task and act executing that task. The process is iterative; it is repeated while the robot is activated (see Figure 3-1 and Figure 3-2). Another feature of the Hierarchical Paradigm is that all the sensed information is used to create a unique world model used by the planner. It is an important issue because the world is based on *closed world model assumption*. This world model needs to define everything within the world. It results in a hard problem not only because the world needs a huge amount of rules, but also because all the possibilities have to be defined. If something unexpected was not considered in the model, the robot could become confused.

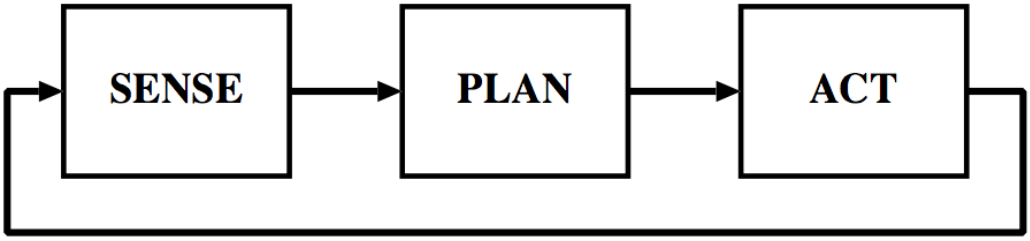


Figure 3-1 : Hierarchical Paradigm Control Flow

The main trouble generated by this paradigm is related with the plan stage. In each cycle the robot updates the world model and plan. Usually, algorithms for these tasks are very slow. In addition to this, the three primitives are executed in order every cycle, which can result in a difference between the world perceived and the real world. Thus, Hierarchical Paradigm has problems to react under unexpected situations or when the environment changes quickly.

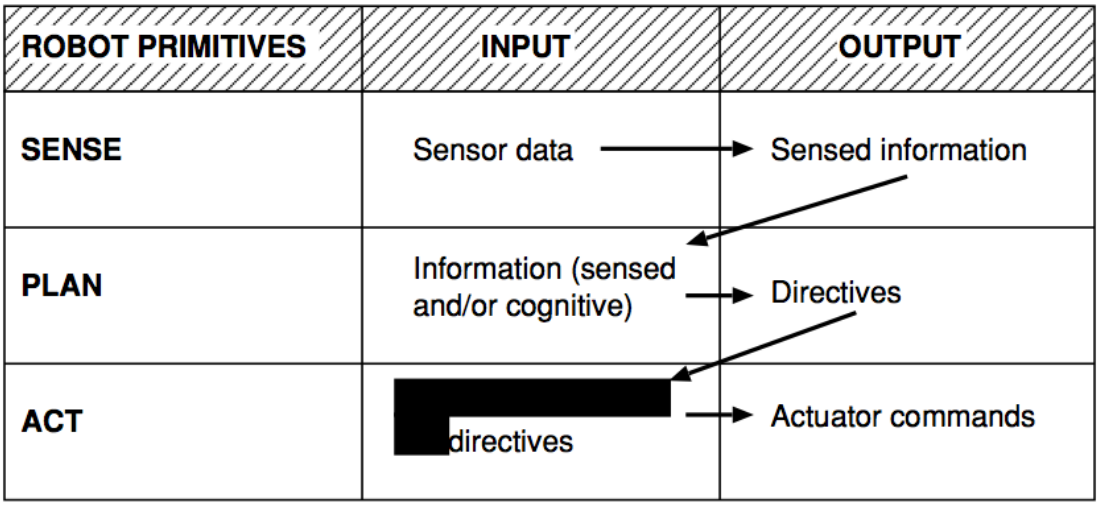


Figure 3-2 : Transactions between the primitives in the Hierarchical Paradigm

3.2.2. Reactive Paradigm

Reactive Paradigm was proposed in order to solve the problems associated with the Hierarchical Paradigm. This paradigm removes the plan stage. It connects directly the "sense" functions with the "act" functions as is shown in the Figure 3-3 and the Figure 3-4. This paradigm was mainly used between 1988 and 1992.

In the Reactive Paradigm, the input of the action comes directly from the information of robot sensors instead of the planner. The sensor is directly connected with the action to

execute. Then, many relations between the information sensed and the action must be defined. These relations are concurrent processes known as behaviors. Behaviors obtain the local information from a sensor and check the best action to execute. This is done independently from the other behaviors. Different behaviors can affect the same actuators in different way as they are independently processes. The resulting action is the sum of all the actions to execute in the same actuator. Consequently, in complex robots is hard to define all relations and adjust all the behaviors appropriately.

Figure 3-3 : Reactive Paradigm Control Flow

The Reactive Paradigm is much faster in execution time. Thanks to this, the robot can take the action immediately after sensors perceive any stimulus. However, without any planner it is hard to manage complex robots and decide in advance future actions. Thus, the Reactive Paradigm is still used on those robots which do not need a planner.

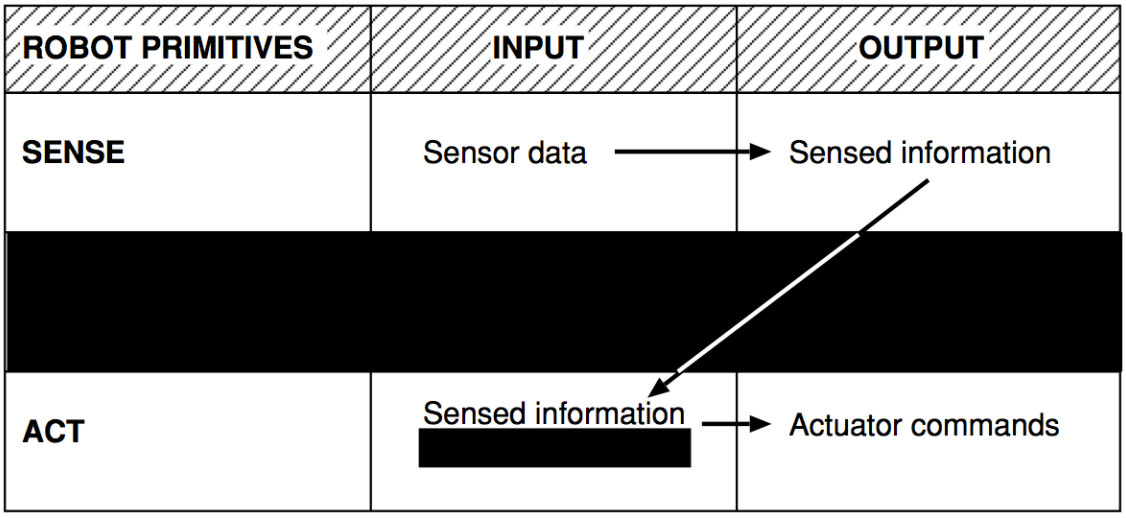


Figure 3-4 : Transactions between the primitives in the Reactive Paradigm

3.2.3. Hybrid Paradigm

Since the 1990's to nowadays, the most used paradigm is the Hybrid Deliberative/Reactive Paradigm. It is a mix between the Hierarchical and the Reactive paradigms taking the better of each one. It consists of two stages. In the first one the

decomposition of subtasks is done by the planner. In the second stage, the perception and acting are done together (see Figure 3-5 and Figure 3-6).

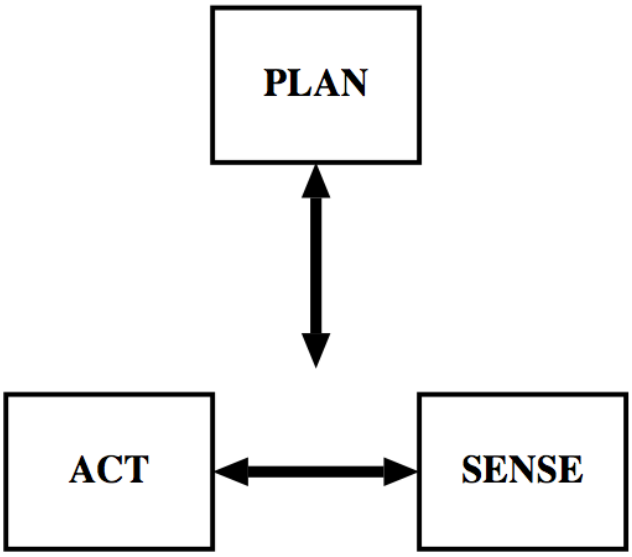


Figure 3-5 : Hybrid Paradigm Control Flow

Therefore, the problem of high computational cost of planner is solved. The planner can be updated hardly ever compared with the reactive behaviors. For instance, the planner can take several seconds doing its computations, meanwhile reactive behaviors are executed many times per seconds. Obviously, depending on the robot features the planner and the reactive part can be adjusted to take more or less important role.

ROBOT PRIMITIVES	INPUT	OUTPUT
PLAN	Information (sensed and/or cognitive)	Directives
SENSE-ACT (behaviors)	Sensor data	Actuator commands

Figure 3-6 : Transactions between the primitives in the Hybrid Paradigm

3.3. Autonomous Unmanned Aerial Vehicles

Aerial vehicles are not especially different from other mobile robots. This type of vehicles has to manage the most common problems for mobile robots such as position

detection, obstacle avoidance or path planning. However, as flight vehicles, they have some singularities to take under consideration. The requirement to keep flying results in a substantial number of issues. In many of these issues aeronautic engineers have been working for decades or even centuries. Nevertheless, in the last years unmanned air vehicles (UAVs) have been taking more significant role and other issues like those related with autonomous flight have appeared or increase substantially its value. Other consequence of UAVs is that new designs of flight vehicles have appeared like quadropters or even insect-like robots. The Table 3-1 [2] shows the most common Micro Aerial Vehicle configurations. As it is shown, each configuration has its own advantages and drawbacks. Depending on the purpose of the vehicle one configuration can adapt to the situation better than others. For instance, blimps solve well the problem of energy efficiency, but their payload/volume rate is an important weakness. Then, this design may be interesting for a vehicle which must keep flying at low speed for long time with limited payload such as doing vigilance tasks. But for other tasks like fast transportation or combat airplanes it would be the choice. That is because airplanes are well designed to flight at high speed, but not at low speed and they require prepared landing strips.

In addition to all issues related with flight control, the autonomous flight represents an important challenge by itself. The autonomy of a robot can be classified in different levels as described in the next section.

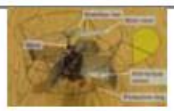
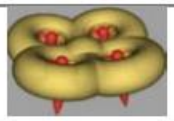
Configuration *e.g.*	Advantages	Drawbacks	Picture
Fixed-wing (AeroVironment)	Simple mechanics, silent operation	No hovering	
Single rotor (A. V de Rostyne)	Good controllability, good maneuverability	Complex mechanics, large rotor, long tail boom	
Axial rotor (Maryland Univ.)	Simple mechanics, compactness	Complex aerodynamics	
Coaxial rotors (EPSON)	Simple mechanics, compactness	Complex aerodynamics	
Tandem rotors (Heudiasyc)	Good controllability, simple aerodynamics	Complex mechanics, large size	
Quadrotor (EPFL-ETHZ)	Good maneuverability, simple mechanics, increased payload	High energy consumption, large size	
Blimp (EPFL)	Low power, long flight operation, auto-lift	Large size, weak maneuverability	
Hybrid quadrotor-blimp (MIT)	Good maneuverability, good survivability	Large size, weak maneuverability	
Bird-like (Caltech)	Good maneuverability, compactness	Complex mechanics, complex control	
Insect-like (UC Berkeley)	Good maneuverability, compactness	Complex mechanics, complex control	
Fish-like (US Naval Lab)	Multi-mode mobility, efficient aerodynamics	Complex control, weak maneuverability	

Table 3-1 : Most commons Micro Aerial Vehicles classified by its configuration

3.3.1. Autonomy Levels

The autonomy level describes how a robot or computer takes and executes the different decisions. The degree of autonomy is inversely related to the degree of human

assistance. Tom Sheridan [3] proposed ten autonomy levels in 1992, from a robot completely controlled by a human to a fully autonomous robot which does not require any type of human interaction. The ten levels are described as follows, being the higher levels the most autonomous.

1. The computer does not offer any assistance, the human does everything.
2. The computer offers different alternatives to the human, and the human choose and execute the actions.
3. The computer selects only few alternatives, discarding the others.
4. The computer suggests a unique action to the human.
5. The computer executes the action if the human allow it.
6. The computer provides to the human an action confirmation time slot and if it does not receive the confirmation within the time slot, the action is executed by the computer automatically.
7. The computer executes the action automatically reporting to the human.
8. The computer reports to the human after an execution only if the human asks.
9. The computer reports to the human after executing an action only when it decides that it should do it.
10. The computer decides about everything and executes all the actions disregarding completely to the human.

The main problem identified in this model is that these scales could not apply to the whole domain. However, they can apply to the different tasks of the domain. With this idea in mind a revision [4] was proposed in 2000 using a model based on a division of four function classes. These classes are information acquisition, information analysis, decision and action selection, and action implementation.

In 2002, the US Air Force Research Laboratory (AFRL) presented the results of a research called "Autonomous Control Level (ACL)" to measure the autonomy level of UAVs. Autonomous Control Level uses a table with a total of 11 levels of autonomy. The ACL table is based on OODA (Observe, Orient, Decide and Act) a division of four classes of functions executed by the UAV. "Observe" is related with the perception and awareness, Orient with the analysis and coordination, Decision with the elections and Act with the action ability.

Few years later, in 2007, a work team from United States funded by the National Institute of Standards and Technology (NIST), developed a framework in order to define the autonomy levels for unmanned systems. This framework was called Autonomy Levels For Unmanned Systems (ALFUS). Several levels shape the framework table using different metrics with smooth transitions between them. Three aspects are measured and classified, Human Independence (HI), Mission Complexity (MC) and Environmental Complexity (EC). Each aspect is classified by several

different factors as shown in the Figure 3-7. For instance, the Environmental Complexity depends on how the terrain is, but also on other factors such the climate and the different objects belonging to the scenario.

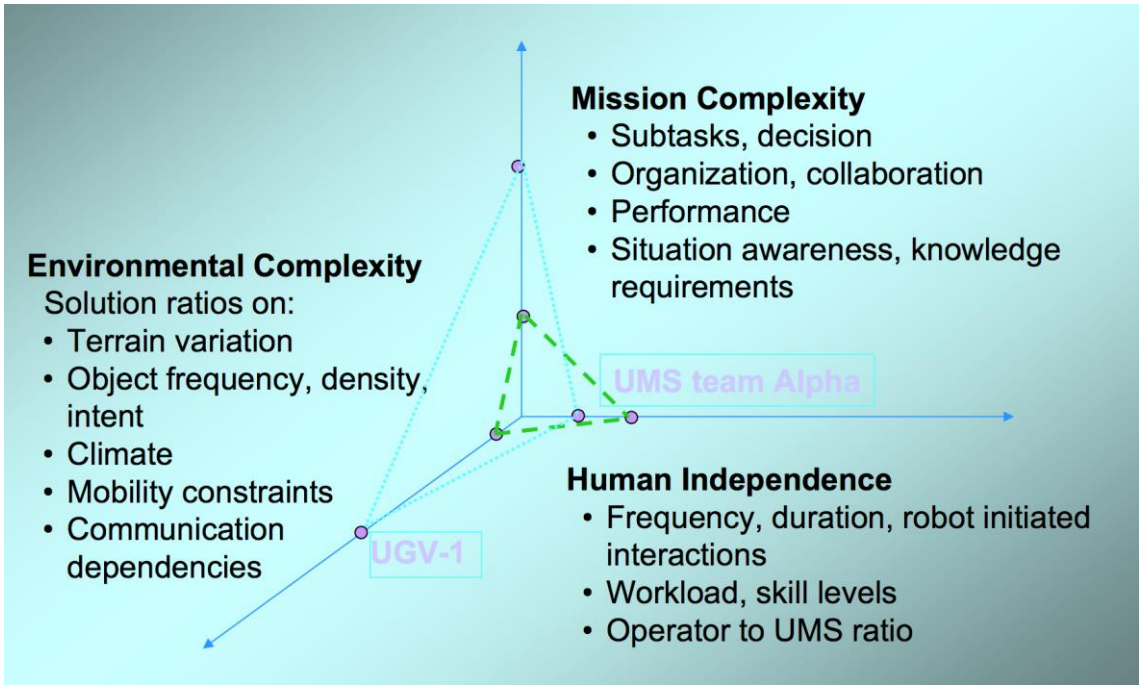


Figure 3-7 : ALFUS Classification Metrics

Autonomous Control Level is applied mainly to big UAVs which flight at high altitude free of obstacles. On the other hand, ALFUS is designed as a generic framework. It takes under consideration any type of UAVs. As ALFUS is a generic framework, specific frameworks better adapted to features of particular type of UAVs are desired. Thus, in 2012, Kendoul [5] proposed the Rotorcraft Unmanned Aerial System (RUAS), a specific classification for the rotorcraft vehicles. These systems flight at low altitude and in different environments and results in a new framework known as Autonomy Levels For Unmanned Rotorcraft Systems (ALFURS).

3.3.2. Autonomy Levels For Unmanned Rotorcraft Systems (ALFURS)

ALFURS is based on the autonomy functions of the robot named as Autonomy Enabling Functions (AEF). These functions are divided into three categories represented by the model GNC: Guidance, Navigation and flight Control. Following the different aspects of each one are explained. To illustrate it better, the Figure 3-8 shows an example of architecture using the ALFURS framework.

- **Flight Control System**. It is in charge of act over the different robot movement systems. Its uses control rules to execute orders with an appropriate output signals

for the control systems. The final purpose is to control the position, speed, altitude and every additional physical aspect related with the flight control of the robot.

- **Navigation System.** Its responsibility is to execute the motorization and to control movements of the robot. To achieve it, the Navigation System obtains status data of the robot and the environment and analyses it. The main target of this analysis is to obtain the "*Situational Awareness*" [6]. Situation Awareness was defined by Endsley [7] in 1988 as "*the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future*". Then, the analysis function gives not only the understanding of the current state of the environment, but also a prediction on how the status may change through the time. To achieve it, the data is obtained and processed for different submodules.
    - **Sensing.** It gathers the different devices on board used for obtain information about the robot or the environment. These devices can be gyroscopes, accelerometers, barometers, push sensors or video cameras among others.
    - **State Estimation.** It receives the data from the sensors. Using the information, it estimates the state of the robot. For instance, it estimates the position of the robot, altitude or speed.
    - **Perception.** It is in charge of creating an environment model using the information from the different sensors. The model produced can include different parts like cartography, object recognition, target or obstacles detection.

- **Guidance System.** This system gathers planning and taking decisions in order to complete goals of the robot. It is the cognitive system which replaces the human operator that would control the robot. It takes the information from the Navigation System. Guidance System uses the information to take decisions which are decomposed in lower level commands for the Flight Control System. Main functions of the Guidance System are described as following.
    - **Mission Planner**. It is the goal generator of the UAV. Mission Planner selects the best action to execute after making an analysis of the scenario.
    - **Path Planning**. Performs the route of the UAV. A sequence of spatial points that the UAV must follow. This path is performed using the navigation information previously stored. The path is generated in order to achieve a specific task. Therefore, the path tries to adapt to the requirements of a specific task. In some tasks the priority could be to find an efficient route, but in other cases could be to find a safe one.

- **Trajectory Generation**. It is in charge of generating the motor functions like the direction of the UAV. The trajectory generation determines weather is possible to reach a specific position according to the UAV physical restrictions. The output information of this submodule can be used directly by the flight controller.

The definition of the autonomy levels of the ALFURS framework is made using the Guidance, Navigation and Control model. The different levels are determined by the complexity of each of the GNC functions of the UAV. In the Table 3-2 all the levels of the ALFURS framework are described. Moreover, it is possible to make a relation between the metrics of the ALFUS and the ALFURS frameworks.

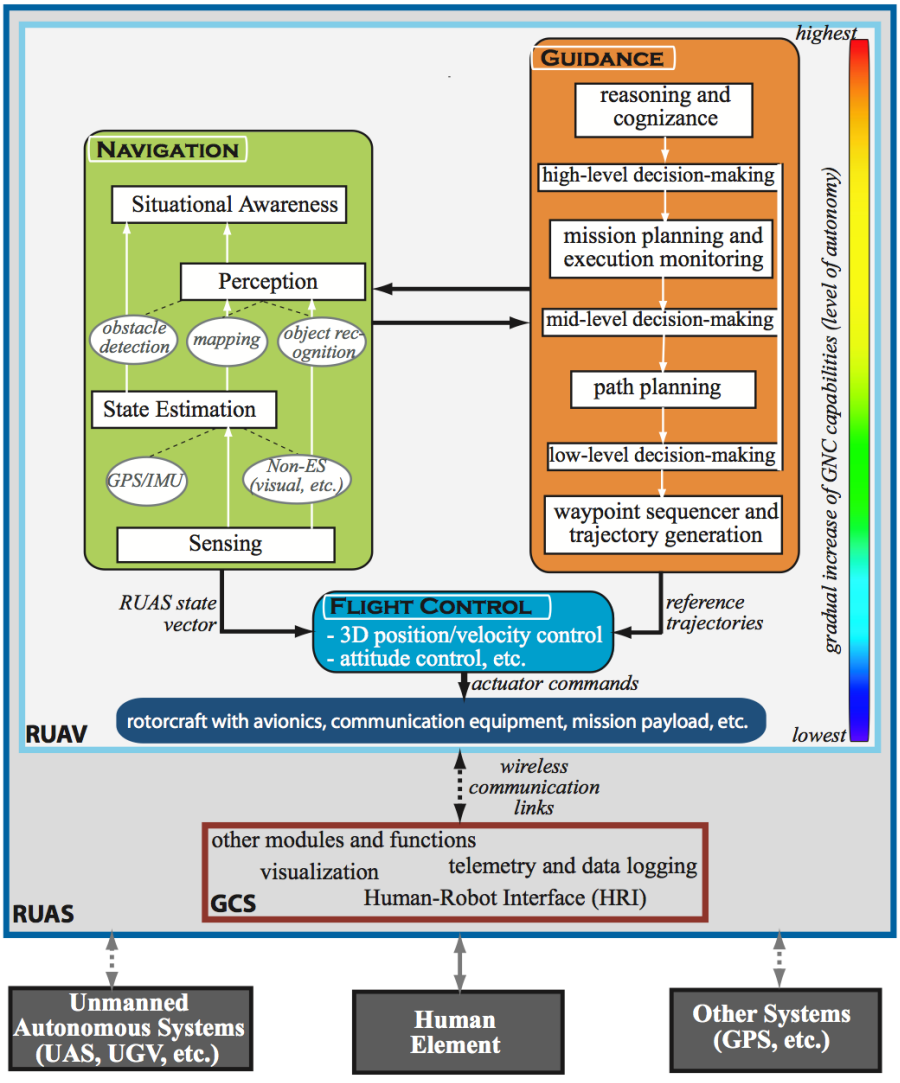


Figure 3-8: UAV's Architecture Model based on ALFURS Framework

Level	Level Descriptor	Guidance	Navigation	Control	ESI	EC	MC
10	Fully Autonomous	Human-level decision-making, accomplishment of most missions without any intervention from ES (100% ESI), cognizant of all within the operation range.	Human-like navigation capabilities for most missions, fast SA that outperforms human SA in extremely complex environments and situations.	Same or better control performance as for a piloted aircraft in the same situation and conditions.	*approaching 100% ESI*	*extreme environment*	*highest complexity, all missions*
9	Swarm Cognizance and Group Decision Making	Distributed strategic group planning, selection of strategic goals, mission execution with no supervisory assistance, negotiating with team members and ES.	Long track awareness of very complex environments and situations, inference and anticipation of other agents intents and strategies, high-level team SA.	Ability to choose the appropriate control architecture based on the understanding of the current situation/context and future consequences.	*high level ESI*	*difficult environment*	*collaborative, high complexity missions*
8	Situational Awareness and Cognizance	Reasoning and higher level strategic decision-making, strategic mission planning, most of supervision by RUAS, choose strategic goals, cognizance.	Conscious knowledge of complex environments and situations, inference of self/others intent, anticipation of near-future events and consequences (high fidelity SA).	Ability to change or switch between different control strategies based on the understanding of the current situation/context and future consequences.			
7	RT Collaborative Mission Planning	Collaborative mission planning and execution, evaluation and optimization of multi-vehicle mission performance, allocation of tactical tasks to each agent.	Combination of capabilities in levels 5 and 6 in highly complex, adversarial and uncertain environment, collaborative mid fidelity SA.	same as in previous levels (no-additional control capabilities are required)	*mid level ESI*	*moderate environment*	*mid complexity, multi-functional missions*
6	Dynamic Mission Planning	Reasoning, high-level decision making, mission driven decisions, high adaptation to mission changes, tactical task allocation, execution monitoring.	Higher-level of perception to recognize and classify detected objects/events and to infere some of their attributes, mid fidelity SA.	same as in previous levels (no-additional control capabilities are required)			
5	RT Cooperative Navigation and Path Planning	Collision avoidance, cooperative path planning and execution to meet common goals, swarm or group optimization.	Relative navigation between RUAS, cooperative perception, data sharing, collision detection, shared low fidelity SA.	Distributed or centralised flight control architectures, coordinated maneuvers.			
4	RT Obstacle/Event Detection and Path Planning	Hazard avoidance, RT path planning and re-planning, event driven decisions, robust response to mission changes.	Perception capabilities for obstacle, risks, target and environment changes detection, RT mapping (*optional*), low fidelity SA.	Accurate and robust 3D trajectory tracking capability is desired.			
3	Fault/Event Adaptive RUAS	Health diagnosis, limited adaptation, onboard conservative and low-level decisions, execution of pre-programmed tasks.	Most health and status sensing by the RUAS, detection of hardware and software faults.	Robust flight controller, reconfigurable or adaptive control to compensate for most failures, mission and environment changes.	*low level ESI*	*simple environment*	*low level tasks*
2	ESI Navigation (*e.g., Non-GPS*)	Same as in Level 1	All sensing and state estimation by the RUAS (no ES such as GPS), all perception and situation awareness by the human operator.	Same as in Level 1			
1	Automatic Flight Control	Pre-programmed or uploaded flight plans (waypoints, reference trajectories, etc.), all analyzing, planning and decision-making by ES.	Most sensing and state estimation by the RUAS, all perception and situational awareness by the human operator.	Control commands are computed by the flight control system (automatic control of the RUAS 3D pose).			
0	Remote Control	All guidance functions are performed by external systems (mainly human pilot or operator)	Sensing may be performed by the RUAS, all data is processed and analyzed by an external system (mainly human).	Control commands are given by a remote ES (mainly human pilot).	*0% ESI*	*lowest EC*	*lowest MC*

Table 3-2 : ALFURS Levels of Autonomy.

Acronyms: ESI (External System Independence), EC (Environment Complexity), MC (Mission Complexity), ES (External System), SA (Situational Awareness), RT (Real-Time).

3.3.3. Research Groups

There are many research groups across the world in the field of autonomous technologies for aerial vehicles. Kendoul [5] made a summary in the Table 3-3 and Table 3-4 including the most relevant ones which work with rotorcraft. These tables include the areas of research and major achievements of each group. However, these tables are limited to research labs and do not include military research groups and industrial companies.

Name of the Group and Institution	Rotorcraft Platforms	Current Research Areas & Projects	Milestones and Major Achievements
Field Robotics Centre, CMU, CMU-FRC-US *www.frc.ri.cmu.edu/projects/landingdemo/* *www.frc.ri.cmu.edu/cademo*	**Class I:** Eurocopter EC135 and the Boeing unmanned Little Bird (ULB) helicopter **Class II:** Yamaha RMAX and R-50. **Class IV**: Otto Quadrotor	- Obstacles detection and avoidance using a 3D LIDAR. - Landing zone detection and safe landing using sweeping 2D LIDAR. - Visual localization and perception.	- Successful obstacle detection and avoidance with RMAX helicopter. - Safe landing area detection and automatic landing with full-scale helicopter .
UAV Research Facility, Georgia Institute of Technology, UAVRF-GTech-US *http://controls.ae.gatech.edu/wiki/uavrf*	**Class II**: Yamaha RMAX and SungWoo Eng. Remo-H helicopters. **Class IV**: Getpsy ducted-fan), GTQ (quadrotor) and GTLama (co-axial).	- Rotorcraft control using adaptive techniques and neural networks. - Vision-based navigation.	- Adaptive autopilot for 3D trajectory tracking and flight control. - Vision-based target tracking. - Vision-based formation flight .
The NASA/Army Autonomous Rotorcraft Project. NASA-ARP-US *http://ti.arc.nasa.gov/projects/apex/projectARP.php*	**Class II:** Yamaha RMAX helicopter.	- 3D navigation in urban environments using vision and LIDAR. - Safe landing area detection (PALACE project). - Mission and path planning.	- Mapping and path planning using a spinning LIDAR. - Vision-based state estimation. - Successful SLAD tests using stereo and LIDAR. - Mission planning for surveillance.
BEAR Group, Berkeley University, BEAR-US robotics.eecs.berkeley.edu/bear/	**Class II**: Yamaha RMAX and R-50 helicopters. **Class III**: Electric Maxi-Joker helicopters.	No current research activities on UAS (to our knowlwdge).	- Flight control using MPC. - Formation flight. - Vision-based landing. - LIDAR-based obstacle avoiadnce.
UASTech Lab., Linkopings University; UASTech-SE *www.ida.liu.se/divisions/aiics/aiicssite/uastech/*	**Class II**: Yamaha RMAX helicopter. **Class IV**: Quadrotors.	- Vision-based landing and localization. - Mission and path planning. - Artificial Intelligence	- Vision-based landing and localization - Path planning. - Mission planning and execustion monitoring.
The French Aerospace Labs (ONERA); ONERA-FR *http://action.onera.fr/accueil*	**Class II**: Yamaha RMAX helicopters.	- Vision-based navigation (target tracking and safe landing). - Mission planning and decisional autonomy (*ReSSAC project*). - UAS cooperation.	- Ground target tracking using vision. - Safe landing area detection using stereo vision. - Mission management.
USL, Univ. of South Florida; USL-USF-US *www.cse.usf.edu/USL*	**Class II:** Yamaha RMAX, **Class III**: Bergen, Raptor 90, Maxi Joker 2	- Flight control. - Vision-based navigation - Fault detection and isolation.	- Automatic flight. - Trafic data collection and analysis.
Kenzo Lab., Chiba University; CHIBA-U-JP *mec2.tm.chiba-u.jp/~nonami* *mec2.tm.chiba-u.jp/uav/main/*	**Class II**: Sky Surveyor, QTW, **Class III**: Hirobo SST-Eagle. **Class IV**: Astech quadrotors. **Class V**: Epson micro Flying Robot (uFR).	- Autopilots design. - Flight control. - Formation flight control. - Vision-based state estimation.	- Automatic flight of different rotorcraft platforms. - Vision-based flight. - Formation flight of two helicopters.
USW@ADFA, Australia *http://seit.unsw.adfa.edu.au/staff/sites/hrp/research/UAV/uav.html* *http://seit.unsw.adfa.edu.au/acme/researchconsult/flight/index.html*	**Class II:** Yamaha RMAX heli. **Class III**: Hirobo Eagle helicopter.	- Flight control. - Optic flow-based navigation. - Automatic landing on a ship.	- Optic flow-based terrain following using the Yamaha RMAX helicopter.
Shenyang Institute of Automation (SIA), SIA-CN *http://uav.sia.cn/en/index.php*	**Class II**: ServoHeli-120 (120 kg), ServoHeli-40 (40 kg).	- Avionics development/integration. - Flight control. - Path planning and UAS cooperation.	- Waypoint navigation with automatic take-off and landing. - Vision-based ground target tracking.
German Aerospace Centre (DLR); DLR-DE *http://www.dlr.de/fl/en/Desktopdefault.aspx/tabid--1377/1905_read-3350/*	**Class III**: ARTIS helicopter (25 kg)	- Vision-based navigation - Stereo vision-based mapping and obstacle avoidance. - Mission management.	- 3D mapping and path planning using stereo vision. - Vision-based flight through obstacle gates.
ARCAA (CSIRO-QUT), ARCAA-AU http://www.arcaa.aero/ http://research.ict.csiro.au/research/labs/autonomous--systems/field-robotics/field-robotics	**Class III**: Vario helicopters. **Class IV**: Quadrotors and Octocopters.	- Robust control of rotorcraft. - Dependable autonomous UAS. - Obstacle detection and path planning using vision and LIDAR.	- Beyond visual range (BVR) flights. - BVR infrastructure inspection. - Mapping and obstacle avoidance using LIDAR and/or stereo vision .
Aerospace Systems and Control Lab., KAIST, ASCL-KR *http://ascl.kaist.ac.kr/*	**Class III**: Voyager GSR 260 helicopter. **Class IV**: Electric T-REX 600 helicopter.	- UAS flight control. - Vision-based navigation. - Obstacle detection and path planning.	- Trajectory tracking and waypoint navigation. - Vision-based flight.
- National University of Singapore, NUS-SG *http://vlab.ee.nus.edu.sg/*	**Class III**: AF25B from Copterworks Inc; Raptor 90 **Class IV**: Trex-450 heli; coaxial rotorcaft.	- Nonlinear control of RUAS. - 3D indoor navigation. - Vision-based navigation.	- Automatic flight control. - Formation flight of a RUAS with a virtual leader. - Vision-based indoor flight.

Table 3-3 : Research Groups working with rotorcraft UAVs (I)

Name of the Group and Institution	Rotorcraft Platforms	Current Research Areas & Projects	Milestones and Major Achievements
- Computer Vision Group, Universidad Politécnica de Madrid, CVG-ES *http://www.vision4uav.com/*	**Class III**: Bergen Industrial Twin and Rotomotion SR20 heli. **Class IV**: Octocopter from Mikrokopter; quadrotor.	- Vision-based pose estimation. - Object tracking. - 3D mapping using vision. - Vision-based flight.	- Vision-based state estimation and ground target tracking. - Visual 3D SLAM. - Visual servoing.
- Robotics, Vision and Control Group , University of Seville, GRVC-ES *http://grvc.us.es/en/*	**Class III**: Heliv and HERO helicopters.	- Control of UAS. - Vision-based navigation and forest fire detection. - Cooperative perception and control of multiple UAS.	- Cooperative detection of forest fire using two RUAS. -Visual odometry and SLAM. - Cooperative faults detection.
- The Center for Advanced Aerospace Technologies (CATEC), CATEC-ES *http://www.catec.aero/*	fleet of heterogeneous UASs (about 6 fixed-wing UASs, 6 unmanned helicopters, and 10 quadrotors).	- Automatic flight control. - Navigation and sense & avoid. - Multi-vehicle coordination. - Avionics and onboard systems. - Automation and robotics.	- European COMETS and AWARE projects.
- Laboratory for Autonomous Flying Robots, TU Berlin, LFAFR-DE *http://pdv.cs.tu-berlin.de/lfafr/*	**Class III**: Aero-Tec CB-5000 heli. (12-16 kg). **Class IV**: Mikado Logo14, self-made quadrotor (5 kg).	- Control of VTOL UAS. - Collision detection & avoidance. - Distributed control of multiple RUAS.	- Load transportation using 3 helicopters. - Sensor nodes deployment by three autonomous helicopters.
- Autonomous Vehicles Group, Aalborg University, AVG-DK *http://www.es.aau.dk/projects/uav*	**Class III**: Bergen Industrial Twin, Vario XLC, Maxi Joker 2, T-Rex 700 Nitro, Thunder Tiger E550. **Class IV**: LMH-120 Corona, T-Rex 450, X-3D-BL quadrotor.	- Flight control with slung load. - Urban UAS navigation. - Task allocation and coordinated flight of AV groups.	- Control of a helicopter slung load system. - Robust control. - Collision-free path generation.
- Autonomous Helicopter project, *http://heli.stanford.edu/* - STARMAC project, *http://hybrid.stanford.edu/~starmac/project.htm*	**Class III**: 90-size XCell Tempest, 90-size Synergy N9. **Class IV:** STARMAC platform with quadrotors.	- Aerobatic and aggressive flight control. - Learning-based control. - Multi-agent control.	- Autorotation-based landing. - Learning-based aerobatic flight. - Bakflip control for a quadrotor. - collision avoidance flight.
- ACFR, University of Sysdney, ACFR-AU *http://www.acfr.usyd.edu.au/research/aerospace.shtml*	**Class III**: UAV vision G18 helicopter.	- Weed monitoring and animals tracking. - Multi-UAV active SLAM. - Cooperative UAS Systems	- Vision-based mapping and classification (fixed-wing). - Aquatic weed surveillance and management (helicopter).
-Aerospace Controls Lab, MIT, ACL-US. *http://acl.mit.edu/*	**Class IV**: Dragonfly quadrotors, AscTech quadrotors.	- UAS modeling and control. - Multi-UAS task assignement. - Multi-vehicle health management	- Indoor flight using VICON. - Persistent mission planning. - Health management of UAS.
- Robust Robotics Group, MIT, RRG-US *http://groups.csail.mit.edu/rrg/*	**Class IV**: AscTech quadrotors.	- Non-GPS indoor navigation. - Planning under uncertainties. - SLAM-based navigation and flight control.	- waypoint guidance and geolocation of ground targets. - Autonomous flight in unknown indoor environments. - Planning for target tracking.
-Autonomous Systems Lab., ETH, ASL-CH http://www.asl.ethz.ch/research/asl	**Class IV**: home-designed quadrotors and coaxial rotorcraft **Class V**: under-development for sFly project.	- AIRobots project (Inspection Rotorcrafts). - sFly project (Swarm of Micro Flying Robots).	- Design of miniature platforms. - Indoor hovering.
- Heudiasyc Lab, University of Technology of Compiegne, HDS-FR *http://www.hds.utc.fr*	**Class IV**: quadrotors and other home-designed multi-rotor configurations.	- Design of multi-rotor configurations. - Modeling and control of RUAS. - Vision-based navigation.	- Automatic hovering. - Vision-based hovering.
GRASP Lab., University of Pennsylvania,GRASP-US *http://alliance.seas.upenn.edu/~kumar/wiki/*	**Class IV:** AscTech quadrotors.	- Autonomous navigation and exploration. - Trajectory generation and control for 3D dynamic environments.	- Aggressive and accurate control of quadrotors using VICON system. - Cooperative manipulation and transportation with aerial robots.
CEA-USI-ANU Collaborative research between France (CEA and USI) and Australia (ANU)	**Class IV:** Home-built quadrotor.	- Nonlinear flight control. - Optic flow-based terrain following. - Visual servoing.	- Automatic flight. - vision-based hovering. - Terrain following in a structured indoor environment.

Table 3-4 : Research Groups working with rotorcraft UAVs (II)

3.4. IARC - International Aerial Robotics Competition [8]

As most of the Robotics competitions, the IARC was created to stimulate the research and progress in the field of aerial robotics. Its first edition was celebrated in 1991 at the Georgia Technology Institute. And now, after 23 years, is the oldest active aerial robotics competition. During this time, IARC has contributed to the evolution of aerial robots technology, from those that hardly can keep flying to modern fully autonomous unmanned vehicles which can interact with the environment.

The international competition has an important status due to its history and achievements. Over the years, the most important universities all over the globe have been participating in the competition with the support of industry and governments. The different missions have been completed and their difficulty grade has been increasing in order to turn into new challenges for the participants.

The IARC competition uses a mission system. IARC proposes one mission designing their rules, requirements and goals to achieve. Then, all the participants, grouped by university teams, try to complete the mission during the annual competition. If they cannot complete the mission in one edition, they can continue working and try to complete the year after. In total, six missions have been completed. Currently, the 7th Mission is being undertaken. All the missions have been designed to become a significant challenge. In the moment of the missions were proposed, the current technology and skills available were not enough to achieve completely the missions. Competition's philosophy can be resuming with the sentence: "*nothing within the World military or industrial arsenal of robots is able to complete the proposed mission at the time the guidelines are released*" as is written in IARC's webpage.

Following, the different missions are explained and the 7th Mission rules are described in detail.

3.4.1. First Mission (1990 – 1995)

The first mission goal was to transport six metal disks which are randomly allocated, from a pick-up area to a goal area. Both areas were separated by a barrier of three feet height which needed to be avoided. The team which transported more disks was the winner.

During the first two years, the different teams were improving their vehicles and finally Georgia Institute of Technology makes the first important achievement. It could make an autonomous flight which included the three stages: take off, flight and landing.

In 1995, three years after the first autonomous flight of the mission, the first mission was completed. The University of Stanford transported one of the disks to the goal. It was the only university which got points and then the first mission winner.

3.4.2. Second Mission (1996 – 1997)

The second mission simulated a toxic scenario where several goals had to be completed. In this scenario there were several drums randomly located which had to be detected by the UAV and then translate their position to GPS (Global Position Satellite) coordinates system. Nevertheless, in order to increase the trouble, the number of drums was not determined. Position and orientation of those drums were selected randomly too and some of them were partially buried. Locate the drums was the first part of the mission. Then, the UAV tried to identify the content of each drum. For doing so, the UAV had to be able to read the labels in the drums. Finally, the last part of the mission was to take a sample, simulated as an orange metal disk which was in one of the drums.

In the first year, a team formed by the Massachusetts Institute of Technology and the University of Boston with the support of Draper Laboratories was the best. They created a fully autonomous vehicle which was able to recognize the whole of five toxic drums and to identify the content of two of them reading the label. However, they did not try to extract the sample.

The year after, the team of the Carnegie Mellon University obtained the best performance. They based his design in the Yamaha R50 helicopter. This model of helicopter was large and powerful compared with most of the other participants. Thanks to this, the team could carry a significant amount of equipment, fuel and batteries which was key point to take an advantage in the mission. The Carnegie Mellon team could identify the position and content of all toxic drums. However, when helicopter tried to extract the sample, it missed for only 2.5 centimeters. The UAV, thought that it carried the sample and came back to the landing area. In the landing area, the aerial robot checked that it did not have the sample and proceeded to try again. It repeated this process until competition time was over. Apparently, the problem was caused by last minute changes in the priority of the processes which resulted in a latency enough to lose accuracy in the position calculation. Finally, Carnegie Mellon team, sponsored by Omead Amidi, was the winner, but they did not complete the entire mission in one day losing the prize of 9000$.

3.4.3. Third Mission (1998 – 2000)

The third mission simulated a catastrophic scenario where the aerial robots had to play a search-and-rescue role. The scenario was large (five acres or more), and there were different obstacles like fires, burning toxic wastes or radioactive spills. Moreover there

were survivors are death bodies represented by animatronics. The goal of the mission was identified the people and if they were still alive and the different hazards in the scenario. The identification included not only the type of hazard, but also the position.

This mission had three editions. In the last edition, in 2000, the German team from the Technische Universitaet Berlin was the winner with its autonomous aerial robot called "MARVIN". The team identified correctly both survivors and death bodies. MARVIN's strategy was to fly high over the obstacles. At high position it searched for bodies and obstacles. When it detected a new one and needed to be closer in order to identify correctly, MARVIN descended vertically avoiding obstacles and got the required information.

3.4.4. Fourth Mission (2001 – 2008)

The fourth mission was designed to fit into three possible scenarios where having an aerial robot would be useful. The first of these scenarios would be a rescue mission with hostages. The UAV would have to enter into the building and take photos of captors and hostages. The second proposed scenario would be to find an archeological mausoleum where some archeologist had died because the building would be collapsing. Once the UAV had found the building, it would have to go inside for taking several pictures and then escape before the building was fallen down completely. Finally, the last scenario would an explosion in a nuclear power plant. The aerial robot would go to the operations building of the power plant and return with images of the control panels.

Despite the mission was designed for three possible scenarios, all the scenarios have similar characteristics. Thus, the mission was in fact a generic version of the three proposed scenarios. The aerial vehicle had to take off from a distance of three kilometers and fly to the goal building. Then, it had to identify the target building among several. When the UAV selects the correct building it has to go inside or send an autonomous sensor to take pictures. Finally, the UAV had to move away to a safety distance of three kilometers. The time for complete the whole mission was 15 minutes.

In 2008, after celebrating eight editions, the fourth mission was considered as completed. However, the judges did not select a unique winner because one of the requirements was not completed for none of the competitors and the conditions of the tournament were to complete all of them. All the parts of the tournaments were successfully completed by different teams, individually and contiguously. Nevertheless, the whole mission never was completed in less than 15 minutes, the stipulated time. Under those results, the judges decided to split the prize between the different teams according to their performance. The team which obtained the best performance was the Georgia Institute of Technology, which also got the prize to the most innovative system.

3.4.5. Fifth Mission (2009)

The fifth mission scenario was based on the navigation inside a building proposed in the previous mission. The main idea is to use an autonomous aerial subvehicle launched from a "*mother ship*" for navigation inside complex interior space composed by hallways, obstacles, small rooms and dead ends. The mission goals were to enter into a building, navigate through the interior, find the target, take pictures of it and return the pictures back to a station located at a certain distance out of the building. To complete it, the autonomous air robots could not use global-positioning navigational aids.

This mission was an important change in the type of vehicles designed for the competition. The size of the robots and scenario were reduced which means new challenges to overcome in the vehicles design. In general, most of the vehicles in the previous missions used a common helicopter design with one main rotor and a tail rotor to avoid torque effect. Since mission number five, the most common configuration has been multiple rotors helicopter like quadcopters or octocopters. Nevertheless, the configuration of the aerial vehicles and their implications were not the only challenge to overcome. Restrictions in size also means less power and significantly less load capability to mount the sensors, batteries and all the systems required to complete the missions.

Mission five was the first, and so far, the only mission completed in the first year. The team of Massachusetts Institute of Technology was the winner. They used a laser system to create a map and an optical system to aid navigation through that map. Despite it needed several tries, finally MIT's team could locate the target, a particular nuclear power plant control panel gauge, took pictures of it and sent back the images.

In this mission, special mention deserves the Embry-Riddle Aeronautical University team, winners of "Most Innovative Air Vehicle" award. They developed a unique vehicle with an impressive design, a "*monocopter*" which had a single rotor blade and an opposing ducted fan.

3.4.6. Sixth Mission (2010 -2013)

The sixth mission was published in 2010. Its purpose was to refine the previous mission behaviors going deeper into the methods to control autonomous aerial robot at indoor environments. Mission six was inspired in a spy situation. The target was to obtain a flash drive memory and get it back. In order to not be discovered by the enemy, the UAV had to put an identical flash memory where the stolen one was. Another hard requirement is that the robot had to be able to read Arabic language writing on the walls, in order to efficiently identify the room where the target was located. Thanks to that, the navigation should be easier in an unknown map and where robots could not use

global position aids, the same as in mission five. Fixed movement and noise sensors were installed in the scenario. The UAVs could disable the movement sensors.

In 2013, after three unsuccessful editions, the mission was completed. The Chinese team from University of Tsinghua was the winner. They used a quadcopter which could locate itself with a few centimeters accuracy. This was possible thanks to algorithms development based on 3D vision.

A significant novelty was introduced in this mission in relation with the organization of the event. The organization decided to hold the competition in two venues at the same time, one in America and another in Asia. That was done in order to make easier to the teams to participate in the competition, especially for Asian teams which normally needed to move to America to participate. Thanks to this action, the competition increased decisively the international scope of the event. Lots of Chinese universities participated in the competition for the first time, but also other countries had representatives for the first time like Qatar or United Arab Emirates. Before it, the competitors have been mainly from United States and Canada, with almost anecdotal participation of European and Asian teams.

3.5. Seventh Mission Description [9] [10]

The last mission until now is mission number seven. It was proposed in October 2013 and in August 2014 [11] the first edition was hold where none of the competitors could complete it. The main goal of this mission is to advance in the autonomous interactions of aerial robots with a dynamic environment. With that goal in mind, the competition has been designed. The UAV must guide several ground robots with semi-random behavior to a goal in order to complete the mission.

Mission Seven introduced a novelty; it is split into two different missions. In "*Mission 7a*" the UAV built by the teams must complete the goals proposed. Once the Mission 7a has been completed, the "*Mission 7b*" will be the next to be held. This new mission will be quite similar, the scenario and the goals will be the same. But it will introduce a new level of interaction because two teams will compete at same time. They will try to complete their goals simultaneously. It means that they have to avoid the other UAV which has an unknown behavior. The mission becomes a two player competition where teams must put in practice the more efficient strategy to win.

Following, a detailed description about the rules of Mission 7a has been written.

The competition arena where the mission is held is a flat square of twenty meters side. In order to provide optical aid, there are white lines of eight centimeters width, creating internal squares in the arena of one meter side. Despite white lines are used to give an

optical aid to the participants, the surface of the arena is unknown. It supposes an important challenge to the participants because they should be able to recognize by vision any surface with no matter which color is or light reflection level has.

In the arena, there are ten ground robots of the model "*iRobot Create*". All the robots have the same autonomous behavior. They are moving following a direction for twenty seconds, then, they spin 180 degrees clockwise and continue moving. However, they do not move completely straight, each five seconds every robot change their direction individually and randomly in a range between five and twenty degrees.

Ground robots also have some behaviors to interact between them and with the aerial robot used by each team. "*iRobot Create*" has a collision sensor in front. When the sensor is activated, the robot turns 180 degrees clockwise. This type of collision can occur when a ground robot collides with another, but also the UAV can use this behavior going down and blocking its way to change the direction of the ground robot. The other method that the aerial vehicle has to interact with ground robots is using the magnetic sensor. In the top of each ground robot a magnetic sensor is located. If the UAV approaches near enough to be detected by this sensor, the ground robot rotates 45 degrees clockwise. The UAV can repeat this maneuver multiple times in order to guide the ground robot in the desired direction.

In addition to the ten ground robots already described, another four ground robots are in the arena acting as obstacles. These four robots are the same model "*iRobot Create*", but they have different behavior and each one holds a vertical cylinder on the top. Each cylinder has a height previously unknown by teams with a maximum of two meters. The cylinder cannot be touched by the UAV, if the aerial vehicle touches any cylinder or any other part of an obstacle robot, it will be eliminated. These ground obstacles robots move in circles of five meters radius from the center point of the arena. However, they do not change their direction when they collide, just stop briefly and continue its route if possible. On one hand, the function of obstacles is acting as moving obstacle to the UAV which must be able to detect them, recognize them as obstacles and avoid any collision with. On the other hand, these obstacles increase the number of collisions with the others ground robots which results in a more dynamic and complex scenario for the mission.

The goal of teams is to guide the ten ground robots out of the arena to a specific side of the square marked with a green line. The UAV must avoid touching the obstacle robots, and must detecting, predicting and interacting with normal ground robots changing the direction of them to achieve the goal. If a ground robot goes out the arena, it will be considered automatically out of the game and will not be able to enter again. However, the aerial vehicle can keep no more than five seconds out of the arena without being disqualified or further than two meters. The arena has also an upper limit which is three

meters. However the vehicle is able to land inside the arena whenever it wants. In fact, it should be a useful way to interrupt ground robots trajectories.

All ground robots, including the obstacles, moves at 0,33 m/s. The speed information, which is fixed and previously known by the UAV, facilitates the complex task of identify and predict movements of mobile targets. However, it is possible that a ground robot suddenly stops working inside the arena and does not continue moving. In that case, it will be considered automatically as a new static obstacle. Nevertheless, these static obstacles can be touched by the UAV without disqualification. Their role is only making more difficult the scenario. This situation could be important because the UAV should take it under consideration in order to identify when it occurs and, then, avoid useless interactions with any "*switched off*" robot.

The arena is located inside a building where it is not possible to use Global Systems Positions (GPS) aids. All the ground robots start in the center of the arena. They are in a circle equally spaced of one meter radius with the center in the central point of the arena. They are facing outward to move away at same time when the mission starts. The obstacles robots are located also in a circle, but having five meters diameter. The UAV starts the mission from one side of the square. The total time of the mission is ten minutes or until all the ground robots have been guided to the goal.

The mission winner is resolved by scoring. The scoring depends on how many robots a team has guided for crossing the green line. To consider that a robot have been guided, the UAV have had to "*touch it on the top*" at least once. If a ground robot crosses the green line by itself without UAV interaction, it will not add any points. At this mission, to complete the mission is necessary to achieve a minimum of seven robots guided. If none of the teams can guide seven robots, the mission will be repeated in the next edition until a team can overcome it. In case of tie in scoring, the team which completes the minimum mission in less time will be the winner. Given that, the mission has an important random component, the teams can try the mission three times per edition. The final score is the best of these three tries.

The aerial vehicles designed by the teams also have physical requirements. UAVs cannot exceed 1.25 meters in any dimension. In spite of the size is limited, there is no weight limit. UAVs must flight autonomously and must provide a system to remote override the propulsion system. The propulsion system must be powered by means of an electric motor using a battery, capacitor, or fuel cell. Teams are allowed to perform a part of the processing on an additional computer mounted in a station outside the arena. The aerial robot communicates with this computer during the mission, but any human interaction is forbidden.

7th Mission introduces several challenges. It has been planned to domain different capacities and strategies like speed, range, object recognition, interaction between robots, robot following, identification and target prioritization, control in landing and descent to a target, knowledge of the environment and mission progress. When all of this stuff will be shown at the completion of Mission 7a, the Mission 7b will start. Mission 7b goes even further. In the future mission, two teams will compete at the same time in the same arena. The scenario will be almost the same, but the strategy will take a main role. The future mission also introduces an UAV of other team will be a huge challenge. In Mission 7a, the behavior of ground robots is previously known, even they move slightly randomly. In the case of interacting with another UAV, movements are not previously known which results in very complex and challenging scenario.

4. SIMULATION PLATFORM [10]

IARC 7th Mission has an important strategy component because the environment is complex and not deterministic due to the random component of the ground robot movements. Therefore, to design a planner which takes appropriate decisions during the mission was imperative to overcome successfully. A virtual real time simulator was created by the UPM team in order to develop the planner which is going to be integrated in the UAV. Recreate the whole mission physically is too hard in time and effort. It needs an appropriate place with a prepared floor following the described rules, 14 ground robots, ten of them taking the role of target robots and the other four taking the role of obstacles. A single try needs lot of preparation, because all the robots must start in a specific position and at synchronized. Moreover, it depends on the right behavior of all the individual components of the UAV as all its controller and perception systems including sensors, actuators and software. As a consequence, the decided solution was to design the planner independently. It was created as a module predefining its inputs and outputs. Thanks to this, the planner could be developed easily without hard dependencies using the simulator as a complete virtual environment. The virtual environment makes easier and faster to develop and test the planner strategies. But also it keeps in mind the final integration with the entire UAV systems.

This section explains how the simulator works and which components will be integrated in the UAV during the competition. At first, the architecture model chosen is described and then, the tools used to develop the entire simulator. At last, the different modules are described, including the contents of the exchanged messages. Special mention deserves the Section 4.1.3.3 which explains the Mission Planner module.

4.1.1. Architecture Design

For the software architecture design, the main feature desired was modularity. It was because the ability to modify or improve the different modules without affecting others was a priority. The model selected was GNC (Guidance Navigation and Control) model. The GNC model is based on the division of the aerial vehicle functions into three well-defined modules. Each of these three components was explained in the Section 3.3.2. The section corresponds with the Autonomy Levels For Unmanned Rotorcraft Systems (ALFURS) and GNC model is used by ALFURS. In addition to these three modules, it was necessary to simulate the competition with all the components. Then, the architecture integrates the following modules: "*Flight Control System*", "*Guidance System*" (which was renamed to "*Mission Planner*" since was more accurate which its occupation), "*Navigation System*" and "*Simulation System*". Under this model, the Navigation System is responsible to create UAV's model of the world, then just a partial model of the entire simulation scene with those elements which the UAV is able

to perceive. Regarding to the Simulation System, its responsibility is to simulate the progress of the whole scene which includes all the robots movements.

According to the architecture model proposed, interconnections between the different modules were necessary. The Figure 4-1 shows these interconnections. The Navigation System was designed to perceive the environment and notify to the Mission Planner and the Flight Controller modules. The Mission Planner had to decide what to do and the Flight Controller had to order the execution of the action.

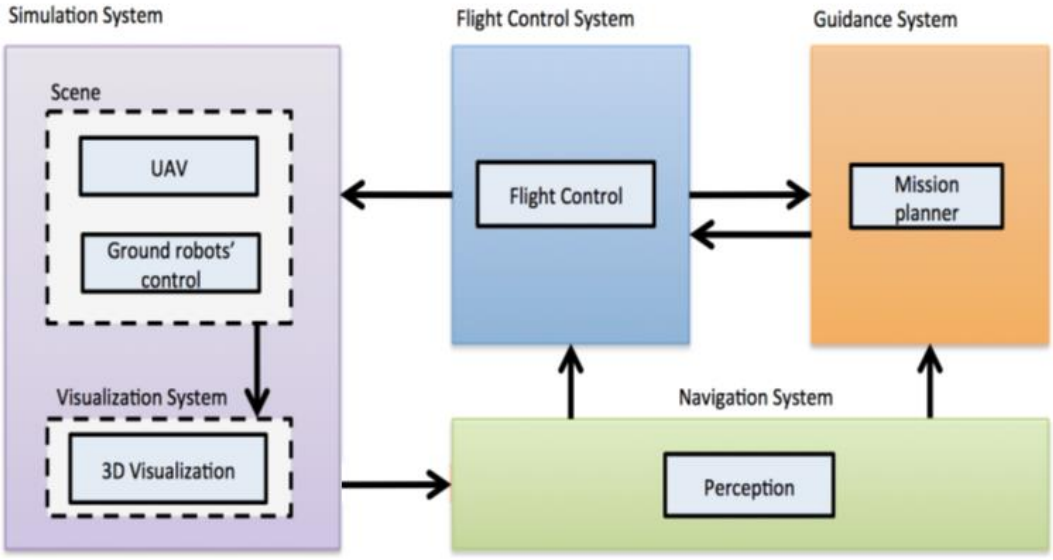


Figure 4-1: Simulator Modules

The architecture designed is based on the hybrid deliberate/reactive paradigm using a similar actions flow. This programming model is highly recommended for many robots purposes and it fits perfectly with the requirements of the UAV for the competition. The autonomous aerial vehicle has to combine reactive actions like avoiding obstacles with higher level planning such as selecting the robot to be guided or deciding the area to start the arena exploration.

The relation between the Hybrid Paradigm and the architecture proposed can be map as following (see Figure 4-2):

- PLAN → Mission Planner
- SENSE → Navigation System
- ACT → Flight Control System

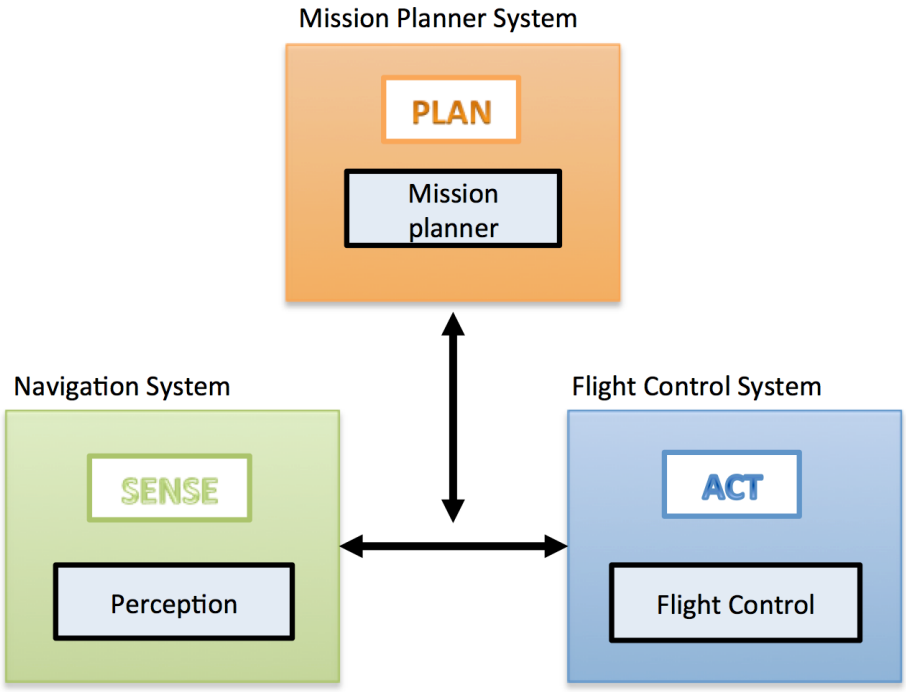


Figure 4-2: Mapping between the Simulator Modules and Hybrid Paradigm

Once the architecture was decided, it was necessary to select the software tools and programming language to be used for its implementation. After an analysis, the conclusion was to use ROS (Robot Operating System). In the next pages, ROS is described in detail as well as the reasons why it was chosen.

4.1.2. ROS – Robot Operating System [12]

Despite its name, Robot Operating System (ROS) is not a pure operating system, but a frameworks collection intended for software development in robotics field. It is called ROS because it provides operating system-like functionality on heterogeneous computer clusters. ROS is widely used and has been employed in all kind of robots: humanoids, autonomous cars, robotic vehicles, mechanic arms, aerial robots, animal-based robots, etc.

ROS project was officially born in 2007 at Stanford University under the name Switchyard. It resulted from the integration of various robotic software frameworks previously prototyped at Stanford like STanford AI Robot (STAIR) and the Personal Robots (PR) program. In 2008, Willow Garage, a robotics research lab and technology incubator, provided enough resources and boosted the project, then the name of the project changed definitely to Robot Operating System. As ROS is based on the philosophy of open source collaborative project and thanks to all researchers and contributors ROS has been increasing his functionality and popularity through these

years becoming a solid, useful, easy to use, adaptable and cheap solution for robotic development.

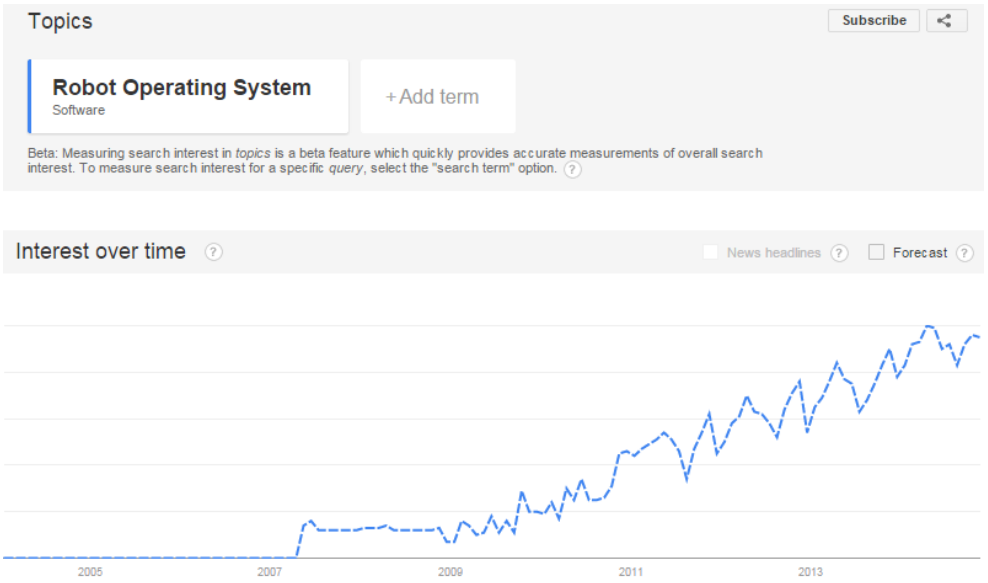


Figure 4-3: Timeline of Queries about Robot Operating System in Google

The different sets of tools are grouped by the term "*ROS Ecosystem*". ROS Ecosystem is split into three different groups:

1. Tools independent from the language and platform for building and distributing ROS-based software. For instance, Rviz is a 3D visualization tool and RQT is a QT-based framework for GUI development and debugging.

2. ROS client libraries. Mainly support C++ (roscpp) and python (rospy), but also for languages like Lisp (roslisp) and in the near future is planning to support Java or Lua, among others.

3. Set of packages which dependencies on ROS client libraries. These packages are created by the ROS supporters and contributors. As the ROS community is growing, more packages are becoming available. The functionalities provided by these packages include hardware drivers, robot models, datatypes, planning, perception, simultaneous localization and mapping, simulation tools, algorithms, among other functionality and useful code.

The main ROS utilities are:

- **Communication infrastructure**. ROS provides communication utilities between processes. ROS processes are called ROS nodes. Its communication system is done

over TCP and the nodes communicate with the master using an XML-RPC protocol. Thanks to nodes abstraction, users can create easily different processes and all the communication between them with ROS interface which takes the role of a middleware layer. The ROS communication interface provides:

- **Message Passing.** ROS offers to the users an easy-to-use message passing system using editor-subscriber mechanisms. Each ROS node can write in a "*topic*" and at same time each node can subscribe to any "*topic*" to receive the messages which are published by the nodes on that topic. It is a many-to-many system of communication completely configurable to easily create the required communication.

- **Remote Procedure Calls.** ROS has the option to create remote services too. It can execute an action when they are called.

- **Robotic Specific Stuff.** ROS has been developed specifically for the field of robotics. This is an important advantage. Although robotics is a very large field where most applications are very specific, there are also many common problems and ways to make robots development easier and faster. Some of the most important are the following.

  - **Robot Standard Messages.** There is a significant quantity of different messages, from the most basics to send an integer or a float number to more complex messages. Additionally, users can create their own messages from existing ones.

  - **Universal Robotic Description Format (URDF)** [13]. ROS has developed a standard to describe physically how a robot is. URDF uses XML (eXtensible Markup Language) to describe its properties, such as sensors, geometry, materials, limitations or joint points between components. Thanks to this language, much functionality has been developed which can be reusable and simpler. The ROS 3D visualization tool, Rviz, understands this format and it is able to show it. And among others, movements or collisions can be used by simulation libraries.

  - **Geometric Robotic Library**. One of the main issues in complex robots as humanoid robots is how to manage easily complex movements when robots have many connected components. In the early days of ROS project, it was detected that implementing it was a painful point for users and important source of errors. To solve it, a library was created with the idea to become a standard for easily managing robotic dependent movements. The library was called "*TF*" [14], transform abbreviation. Thanks to TF library, users can

define how the components are connected using transforms and define complex movements through the time in easier and standard way. Thus, simplifying significantly the development of every type of transformations, from static to dynamic ones. A typical application is the control of a complex robot arm. This library can understand URDF format if the components are properly described with their translations and rotations.

  - **User libraries**. ROS has an important community and thanks to its BSD licenses and collaborative philosophy is growing and getting advanced and useful libraries in all the different subfields of robotics.

- **Set of tools**. ROS provides to the users tools to help them debugging, tracking and visualizing their applications. The different tools are launched by command line. The most important of this tools are two:

  - **Rviz: a 3D visualization tool.** It works as a server receiving messages with the information of the scene and sensors in a specific format. Rviz shows the scene in 3D and allows multiple options like moving the camera around the scene. Moreover users can visualize useful information such as the sensors data or the time elapsed since the simulation has started. Lastly, it is important to know that Rviz allows creating new display capabilities adding plugins. Users can develop their own plugins and use them.

  - **RQT**. It is a framework based on the QT library. Some of RQT tools are very useful for debugging purposes. It is possible to see all the information which is sent into messages with "*rqt_topic*", send a message with "*rqt_publisher*" or get the graph which represents all processes and communications between them with "*rqt_graph*".

ROS has a modular architecture based on nodes. Each node is launched and in the properly device with a service called "*ROS Core*". This architecture is appropriate to robots applications where often there are a significant amount of specific controllers and drivers capturing information. It provides a simple way to allocate the nodes to execute them. And, at the same time, a powerful communication service, which is critical for these robotics heterogeneous distributed environments.

There are many other good operating systems solutions for robotics software development. There are specific ones in the field of robotics like RROS (RobotBASIC Robot Operating System), but there are also general purpose solutions like QNX or VxWorks. However, ROS fits perfectly with the IARC 7th Mission project and it was selected to be used by the UPM team for several reasons.

First of all, ROS is widely used on a variety of successful robotic projects. It means that ROS has been tested and improved by many projects. It has a large community that has previously addressed those issues you find when you are developing. The community and ROS support group not only can help you, but also they have created a significant amount of good documentation.

The second reason because ROS fits with our project is its collaborative philosophy. Their free BSD licenses do not result in any additionally software cost when you are developing with ROS. At same time, as open source project, it is designed to simplify the way of making your contributions to the community. In addition its large community has produced a significant amount of contributions of all flavors.

Another important reason is ROS modular architecture. It provides a simple communication interface and easy way to allocate the different nodes. This is very useful for our developing method, because it is possible to develop a simulator and to migrate its nodes to the real system just allocating the nodes in the UAV or to the auxiliary computer.

Finally, ROS includes debugging tools and a 3D visualization system. Rviz is a great option to the 3D visualization of our simulator, saving the effort of creating a 3D visualization system from scratch or with more complex tools. On the other hand, the debugging tools can show and log all the information in real time, but its presentation and usability is too basic. Although it requires more effort, you can use the services of these tools to create your own scripts or more complex debugging tools adapted to your project requirements.

For developing an IARC 7^{th} Mission Simulator and the planner which takes the decisions of that mission, Python was chosen as programming language. In the beginning of 2014, when the simulator development started, only the support of client libraries of C++ (roscpp) and Python (rospy) were advanced enough to take these two languages under consideration. Thanks to ROS modularity architecture, it is not necessary to write all the modules in the same language. For instance, it is possible write some nodes in Python, others in C++ and others in any other language having a ROS client library. It avoids the significant restriction which supposes to use a specific language and it makes a project highly adaptable. In our case, there was a significant amount of reusable code in C++ for control the UAV and robot devices. But thanks to ROS modularity, it was not necessary use C++ for the entire development. Instead of C++, Python was chosen for the simulator and planner due to the development features of them.

Python is a high level language with a clean syntax. It allows producing readable and multiplatform code. It is highly recommendable to a large amount of projects supporting

Object Oriented Programming (OOP), imperative programming and functional programming. For these reasons Python has become one of the most used programming languages in the last decade, but C++ is still significantly more used than Python. The Python's main advantage is due to its modern and powerful features like high level data types or dynamic typing. Thus, Python's programs are usually shorter and its development process faster than most of the general purpose languages, including C++. The C++ language is suggested to use when high performance is desired and also for big projects. As the simulator is composed of several nodes which are not big enough to justify the use of C++, Python should be more convenient. The performance of Python for the simulator features was enough. But in any case, the modularity of ROS and the possibility to execute C++ code inside Python can lead to future performance improvements if required.

ROS has several distributions. When the project started the recommended version was the last "*ROS Hydro Medusa*". Following the recommendations it was the chosen distribution for the project, running under Linux Ubuntu 12.04. Linux Ubuntu 12.10 and 13.04 are also supported operating systems for the ROS Hydro Medusa version, nevertheless other operating systems such as Windows, Fedora or Mac OS X are only at experimental stage.

4.1.3. Simulator Implementation

After architecture was decided and tools for implementation were chosen too, the simulator was designed, coded and tested. In this section, the different modules of the resulting system are explained in detail as well as the communication between the different modules and the content inside the different messages.

4.1.3.1. Architecture Overview

The simulator architecture is based on Hybrid Deliberate/Reactive Architecture. Then, it has three main components for the aerial autonomous robot simulation. In addition, another component is used to simulate realistically all the competition environment. This component is called the Scene Simulator and it simulates the whole scenario during the time of a single try on the competition. This simulation includes all the interactions between the different agents such as collisions, ground robot movements or interactions with the UAV, always according to the mission description from the rules. Finally, a last component generates 3D models of the arena and shows it in real time. This last module is optional. The full simulation can be executed without this 3D visualization component. However, it is really useful for all the developing process. It helps to understand want is happening in the scene and detect possible errors or useful information about the mission.

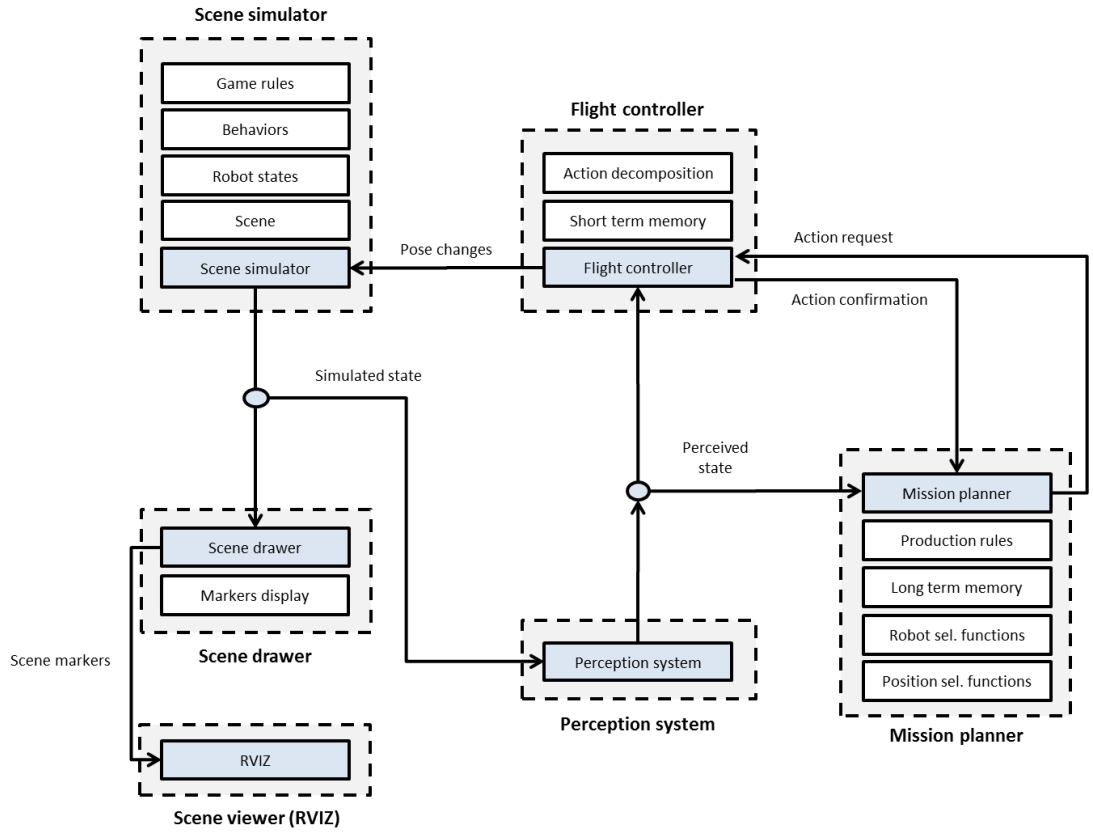


Figure 4-4: Flight Simulator Architecture

The Figure 4-4 shows these five modules and the interactions between them via messages. In order to create more structured code, the functionality of each module is divided in different files according its purpose. Each module has a main file in charge of communication and calling auxiliary functionality in the other files. Auxiliary functionality is distributed in different files in order to avoid big files and to simplify searching into code. For instance, the Flight Controller has three files, the main file and two auxiliary files. Short time memory and action decomposition files are called by the main file when required.

The simulator architecture works as a virtual reality environment. It works independently from all hardware components and the real environment. In the Figure 4-5, the real architecture is shown. It shows how the components developed with the simulator will be integrated with the UAV architecture. In the UAV architecture, the Mission Planner component is the deliberating module. Mission Planner takes the information from the Navigation System, selects a decision according to that information and notifies it to the Reactive component. Thus, Mission Planner is not hardware dependent and will be allocated in the real UAV during the mission. However, the Reactive component, called Flight Controller, is hardware dependent. It depends on

actuators to control the UAV movements. Then, it will need to decompose the general UAV movement instructions generated into low level signals for the UAV actuators such as engines or any flight control surface. Finally, the Perception System of the UAV depends completely on the information from the sensors. It will need to take the information form the sensors and create its perception state of the environment. Therefore, both low level part of the Flight Controller and Perception System will be integrated with the Mission Planner and the high level Flight Controller for the final architecture.

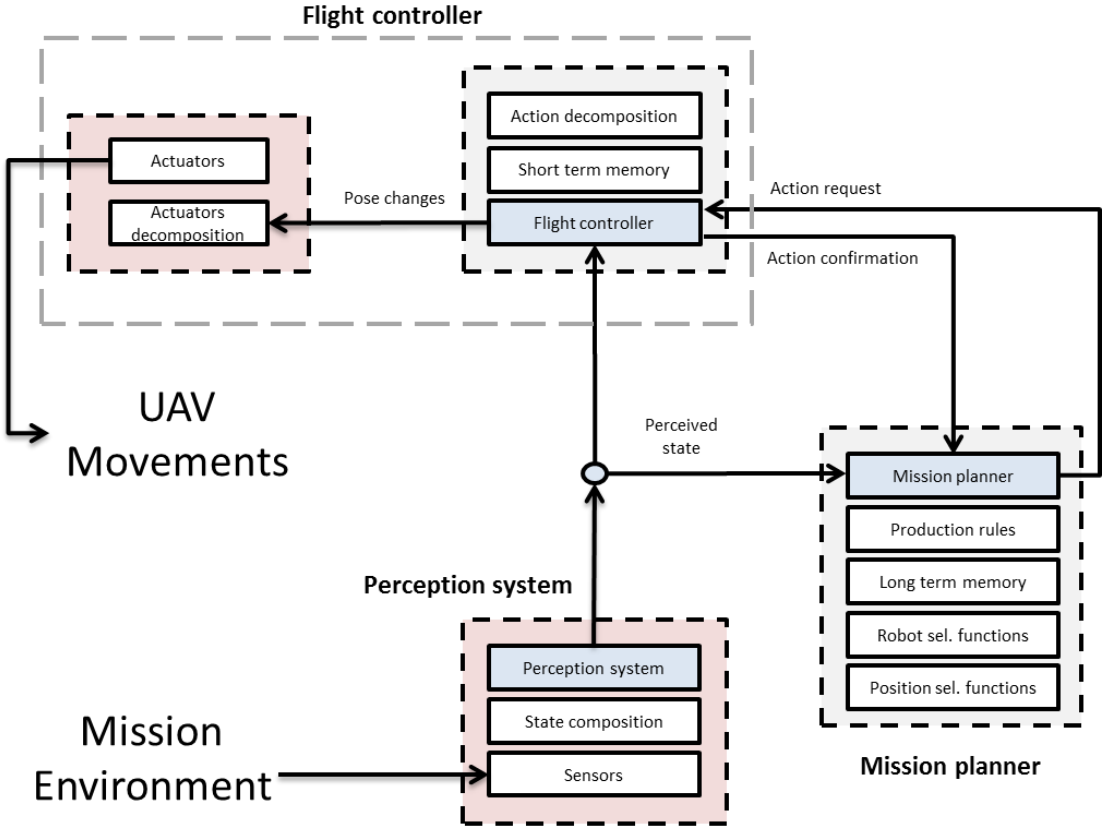


Figure 4-5: Aerial Vehicle Architecture

4.1.3.2. Internal Control Architecture Messages

As described, ROS is strongly addressed to create different independent nodes and communicate them using the editor-subscriber method provided. Acting as middleware, ROS simplifies the use of the systems communication. What users have to do is, first of all, to define the connections or "*topics*". Then, decide who is going to be the editor or publisher of each connection. And decide who is going be the reader or subscriber of each connection. Finally, define the type of messages for the different connections. Focusing in the last task, ROS allows users to create different messages using from

simple data types, to arrays and composed types. Thanks that ROS allows to build your own composed types is possible to create complex data types and manage them. Moreover, ROS creates a class in the programming language you are using, Python in the case of this simulator, to manage the information and make the conversion. Thus, users do not need to use message types, but equivalent classes.

For the communication of the entire simulation system a total of five messages and three additional auxiliary messages data types were created. These are all the originally existing types used in the system:

- **bool**: A simple boolean with binary value, represented by 1 bit.
- **uint8**: Numerical type. Unsigned integer of 8 bits.
- **int8**: Numerical type. Signed integer of 8 bits.
- **uint16:** Numerical type. Unsigned integer of 16 bits.
- **int16**: Numerical type. Signed integer of 16 bits.
- **uint64**: Numerical type. Unsigned integer of 64 bits.
- **float64**: Numerical floating type of 64 bits.
- **Point**: Complex type defined by ROS in a directory called geometry_msgs. It is a 3 dimensional point with coordinates x, y and z.

The 3 auxiliary messages data types were created following the principles of encapsulation making easier to create, understand, manage and maintain all the messages information. The auxiliary data types created are the next:

- **UAVState.** The current position of the aerial vehicle is described in this message. It has several fields.
  - **id** (uint16). Identifier for the UAV. This field is not necessary for Mission 7a where only one aerial robot is on the scene. However, in the future Mission 7b two aerial robots will be under same arena at same time and unique identifiers will be required to manage the scene.
  - **position** (Point). Current position of the UAV in the map, using a 3 dimensional point.
  - **direction** (float64). Orientation of the UAV. The UAV can rotate only over the vertical axis.

- **TargetRobotState**. Ground target robots have their own message to describe its position and state in a specific instant. For describing it, some fields are needed.
    - **id** (uint16). Unique identifier of each ground target robot.
    - **position** (Point). Current spatial point where the ground robot is in the arena.
    - **direction** (float64). Orientation of the ground robot. As it only moves in 2 dimensional, the complete physical representation can be described only with a point and direction.
    - **state** (uint8). Target robots behavior can be described with a state machine. They have several states: wait, run, collision, noise, reverse and touched. The current state uses an integer which is taking the role of an enumeration type.
    - **tfreverse** (float64). Ground robots turn 180 degrees each 20 seconds. This field described the time elapsed since the last time the robot execute its reverse movement.
- **ObstacleRobotState.**
    - **id** (uint16). Unique identifier of each obstacle robot.
    - **position** (Point). Current spatial point where the obstacle robot is in the arena.
    - **direction** (float64). Orientation of the obstacle robot. Similar to target robots. The orientation of the robot and a point is enough to determine its entire position.
    - **state** (uint8). Obstacles have three states: wait, run and collision. These states are enough to simulate correctly the entire obstacle ground robot behavior.

The messages between the different nodes are built using both predefined and the specific robot state types. There are the 5 resultant messages interchanged by the different nodes of the system.

- **Simulated State**. Message sent by the Simulation System for sharing the current state of the whole scene of the mission, including all the robots states, the UAV and the simulation time. The complete data has the following fields.
    - **time** (float64). Time elapsed since the mission begins.

- **targets_in** (uint8). Number of current ground robots that have crossed correctly guided by the UAV the green line.
- **targets_out** (uint8). Number of current ground robots which have abandoned the arena except those which crossed the green line.
- **terminated** (bool). A simple sign to notify if the competition has finished because the 10 minutes of time are over, or if all ground robots are out of the arena.
- **succededAt** (float64). Minimum number of target ground robots correctly guided to the goal. According to the mission rules, a minimum of 7 ground robots must be guided to the goal to accomplish the mission.
- **targets** (TargetRobotState[ ]). A list of the current status of each target robot.
- **obstacles** (ObstacleRobotState[ ]). A list of the current status of each obstacle robot.
- **uav** (UAVState). The current state of the UAV.

- **PerceivedState**. Message sent by the navigation system to the flight control and mission planner system. It is a subset of the simulated state. It includes the robots which are in the vision area of the UAV. Explicitly, these are the fields:
  - **time** (float64). Time elapsed since the mission begins.
  - **targets** (TargetRobotState[ ]). A list of the current status of each target robot seen by the UAV.
  - **obstacles** (ObstacleRobotState[ ]). A list of the current status of each obstacle robot seen by the UAV.
  - **uav** (UAVState). The current state of the UAV.
- **ActionRequest.** Message sent by the mission planner to the flight control system in order to notify what to do next. The message has several fields:
  - **id** (uint64). Unique and sequential identifier for the request action.
  - **type** (uint8). Action chosen by the mission planner to be executed by the flight controller. It can take different values: find, touch, get close, land back, land front and hover. This is explained in detail in the Section 4.1.3.5 related to the Flight Control System.

- **robotid** (int16). It is used as parameter in some of the action types, when the action needs to identify a specific robot.

- **position** (Point). It is used as parameter in some of the action types, when the action needs to identify a point. For instance, it is used to find a robot in a specific point. The aerial vehicle moves to that point and look for robots in its way.

- **ActionConfirmation.** This message is sent by the flight controller to the mission planner. It notifies when the flight controller has finished its action. It informs if the action has been completed successfully or not with additional information which helps the mission planner to take further decisions.

    - **id** (uint64). Unique and serial identifier. It is used to notify what action has been finished.

    - **status** (uint8). The reason because the action has been finished, in normal case because it has been completed successfully.

- **PoseChanges.** This message is sent to the scene simulator system by the flight controller. It is a request of movement for the UAV. The simulator takes this information to update the position of the UAV.

    - **dx** (float64). Movement request in the coordinate x.

    - **dy** (float64). Movement request in the coordinate y.

    - **dz** (float64). Movement request in the coordinate z.

    - **dir** (float64). Request to turn in the vertical axis.

4.1.3.3. Mission Planner

The Mission Planner module is in charge to decide the next action to be executed by the Flight Controller. It takes the decision according to the information received from the Navigation module.

The Mission Planner generates long term goals for reaching in around five or ten seconds, depending on the action. The mission planner was designed using a strategy named as "*Select and Guide*". In "*Select and Guide*", the UAV first selects a ground robot and then guides the selected ground robot to the win line. To achieve it, it interacts with the ground robot by touching it when required. However, in any moment, the UAV can decide to unselect the ground robot or select another. For example if the

UAV find an obstacle in its trajectory, it could decide that would be better to select another robot.

The Mission Planner must deal with complex environment. It only knows the current situation of the area seen by its range of vision. The rest of the arena is changing constantly. Mission Planner only can predict or guess where the different robots would be located, using the information obtained by its sensors. Mission Planner have to solve questions like "*Which robot should select the UAV?*" Or "*Under what circumstances should it abandon a robot previously selected?*" Or "*Where should it look for ground robots if the UAV cannot see any robot to guide?*" These are only some of the questions which determine the mission planner strategy. The goal of the Mission Planner is to take the best possible decision after making an analysis using the world model and its long term memory.

Mission Planner uses as input the world model perceived by the Navigation System. The content of the model include all the robots information in the vision area, the UAV's own position and the time elapsed since the mission began. Ground robots information, whether they are obstacles or not, includes the spatial position, orientation, state and, only in the target ground robots, time left to its 180 degrees turn. In addition, Mission Planner has a long term memory where it stores all the information which could be relevant for taking decisions. For instance, it can store when and where it has seen the different robots, the number of robots already guided to the goal or which areas have been already visited. Long term memory improves the knowledge about the environment and if used appropriately results in better strategies.

Mission Planner works receiving the input and generating an action. An action is a specific goal which can be understood and reached by the Flight Control System. The action is sent to the control system. And then, Mission Planner changes its own state to "blocked" until the action has been executed. When the action is executed, Mission Planner will be called again looking again for the best available actions.

Mission Planner had been implemented as a hierarchical planner. Figure 4-6 shows partial decomposition of the implemented hierarchy. On the top, "*HerdRobots*" is the general goal of the UAV, guide robots to finish line. It can be performed by a set of simpler goals of the type "*Guide*". "*Guide*" action guides a specific robot located in point of the map to the finish line. At same time, "*Guide*" can be divided in lower level goals. "*GetClose*" and "*Touch*" are actions at this lowest level which are executed by the flight control. The description of these actions is detailed in the flight control section. The hierarchy planner is a recursive process because in a low level may exist goals also present in upper levels.

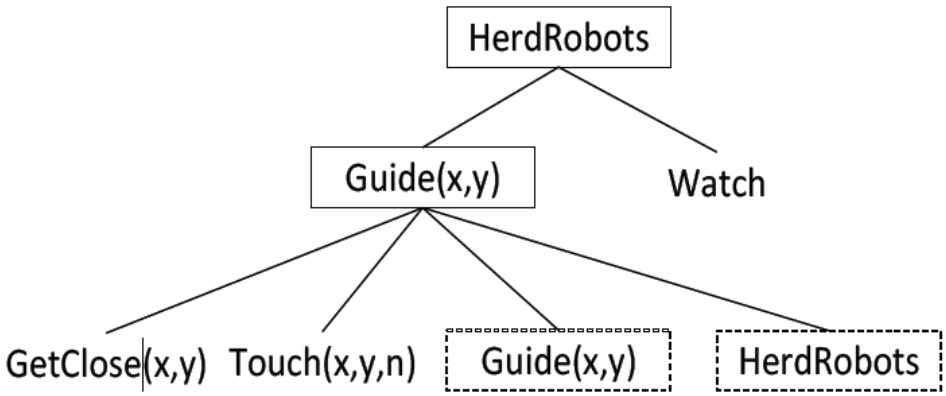


Figure 4-6: Partial Decomposition of the Hierarchical Planner

The Mission Planner implementation is in a ROS node called "*mission_planner*". This node communicates with other nodes. Specifically, it receives messages from the Navigation system, and not only receives from, but also sends messages to the Flight Control System as is described in the Figure 4-7. The messages are:

- **PerceivedState**. Received from the Navigation System. It contains the world model created with the information of the different UAV's sensors.
- **ActionRequest**. Sent to the Flight Control System. It contains the action decided by the planner to be executed by the UAV.
- **ActionConfirmation**. Received from the Flight Control System. This is a message used by the flight control system to notify that it has executed its action and the planner should look for a new one.

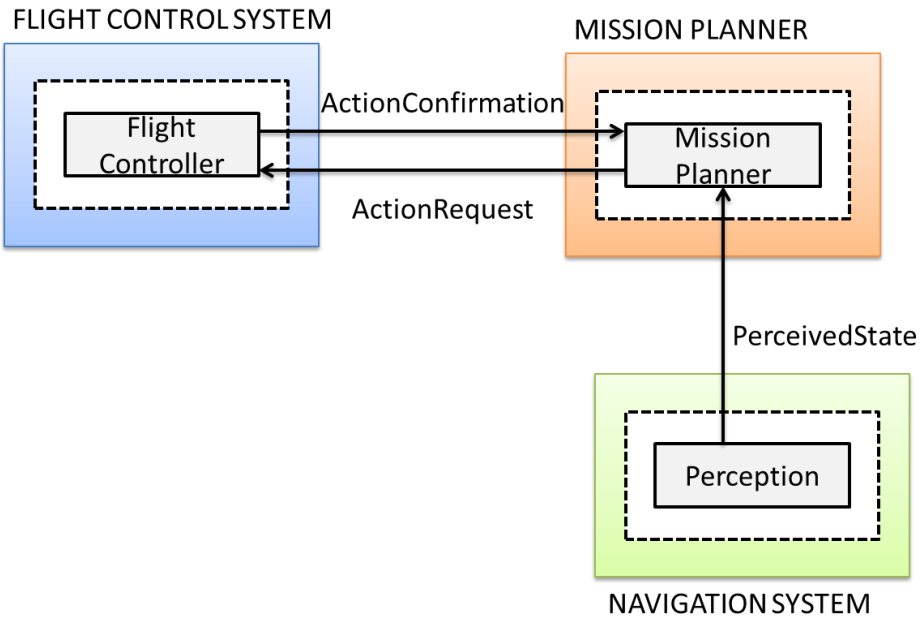


Figure 4-7: Mission Planner Communications

4.1.3.4. Navigation System

The Navigation module is in charge to take all the information from the sensors of the UAV. Then, it creates the world model and sends it to both the Mission Planner and Flight Control systems.

In the built simulator, the Navigation System is a very elementary module. It receives the whole scene as input and selects the information that should be perceived by the aerial robot. Specifically, the perception uses a fixed vision range. The world model created by the Navigation System includes the UAV and all of the ground robots in its vision range. The vision range is determined by the hypothetical range of UAV's sensors. At this point in time, according to the sensors experiments the UAV vision when it is flying at 1.5 meters of altitude is a circle of four meters radius behind the UAV.

However, the final navigation system is not as simple as this one created only for the simulation. Perception is one of the most complex tasks in robot development. It needs to collect all the data from the sensors and translate into a world model. This procedure is difficult because depending on sensors features, the translation can be more or less complex and more or less accurate. There are many challenges in IARC 7th Mission related with perception. For instance, obstacles recognition among all the ground robots or how to determine what direction a robot is moving for. Perception is a complex task. It needs to use hardware sensors. Then, it requires complex data analysis like image

recognition. With high level of complexity and depending on sensors, the Non-Virtual Navigation System module perceives the world with accuracy errors. Thus, the Virtual Navigation module created in the simulator is unrealistic, despite it works enough well for the simulations. These are important facts to take under consideration for the final integration in the UAV.

As well as the other modules, Flight Control System was implemented as a ROS node. The node was called "*perception_system*". It receives the whole scene of the world from the simulation and sends its world model to both Mission Planner and Flight Controller systems as it is shown in Figure 4-8. Message types are described as following.

- **SimulatedState.** Received from the Scene Simulator. It has the entire current scene of the world previously simulated.
- **PerceivedState**. Sent to the Mission Planner and Flight Controller. It has the robots information perceived by the UAV's sensors.

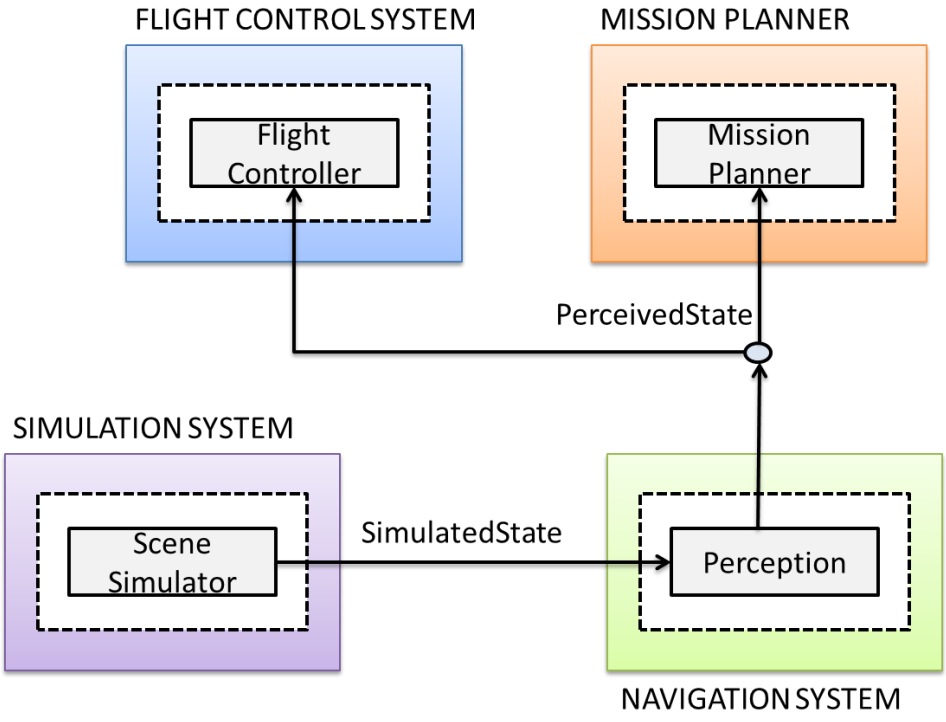


Figure 4-8: Navigation System Communications

4.1.3.5. Flight Control System

The Flight Controller, also implemented as ROS node, is in charge of the execution of actions demanded by the Mission Planner. The action received is at upper level. The mission of Flight Controller is to decompose each action in lower level instructions and

execute them. The lowest abstraction level is hardware instructions. For instance, change the power of UAV's engines to descend. However, instructions of the lowest level are not implemented in this simulator because they are hardware depending. The output of the flight control is simpler because its goal is to develop the Mission Planner and obtain valuable information about strategies. Then, Flight Controller only sends the displacement of the UAV to the simulator for updating its position.

The action to execute sent by the Mission Planner can take several values, described as follows.

- **Find**. This action is designed to find new robots. The UAV moves to the point in the arena decided by the Mission Planner. This action will finish if the UAV found a ground robot while going to that point or when it arrives to the point.

- **Touch Robot**. It is in charge of touching the upside of the robot where the magnet sensor is located. The message includes the position of the specific robot to be touched. This function works as an internal state machine with the relative position of the UAV and the position of the target ground robot. Different machine states are described as following.

    - **Reaching**. The UAV approaches to the target ground robot keeping its flight altitude.

    - **Descending**. The UAV moves down approaching to the target robot. It approaches enough to be detected by the magnetic sensor of the ground robot.

    - **Ascending**. When the UAV have been detected by the robot sensor, the UAV return to its default flight altitude moving up.

- **Get Close**. When this action is under execution, the UAV follows the selected ground robot. This action is similar to the "Reaching" phase of Touch Robot action.

- **Land Back**. This action is for landing close to the target robot. The goal is to land back near ground robot, close to the place where it is going to do its 180 degrees turn. Then, when ground robot came back, it collides with the UAV and turn back again. That action results in guiding the ground robot in a direction. This action is quite similar to Touch Robot action, but little more complex. Landing has reaching, descending and ascending states too. But, it has more complex computation because it needs to predict where the robot is going to turn and select an acceptable landing point. Also it is possible that the landing zone

was occupied. In that case, the UAV cancel the action and notifies to the Mission Planner about the situation.

- **Land Front.** This action is similar to the land back, but selecting the landing point in front of the current direction of the ground robot.

- **Hover.** It is used to keep the UAV flying at same position. The UAV does not move to any direction and does not change it position.

The ROS node is named as "*FlightController*". It receives input from the navigation system and sends appropriate commands to the simulator in order to move the UAV. It has bidirectional communication with the mission planner as shown in the Figure 4-9. The messages are described as following:

- **PerceivedState**. Received from the Navigation System. It has the world model created with the information of the different UAV's sensors.

- **PoseChanges**. Sent to the Simulator System. It has the movement orders for the UAV.

- **ActionRequest**. Received from the Mission Planner. It has the high level action decided by the planner.

- **ActionConfirmation**. Sent to the Mission Planner. It is used to notify when an action is finished.

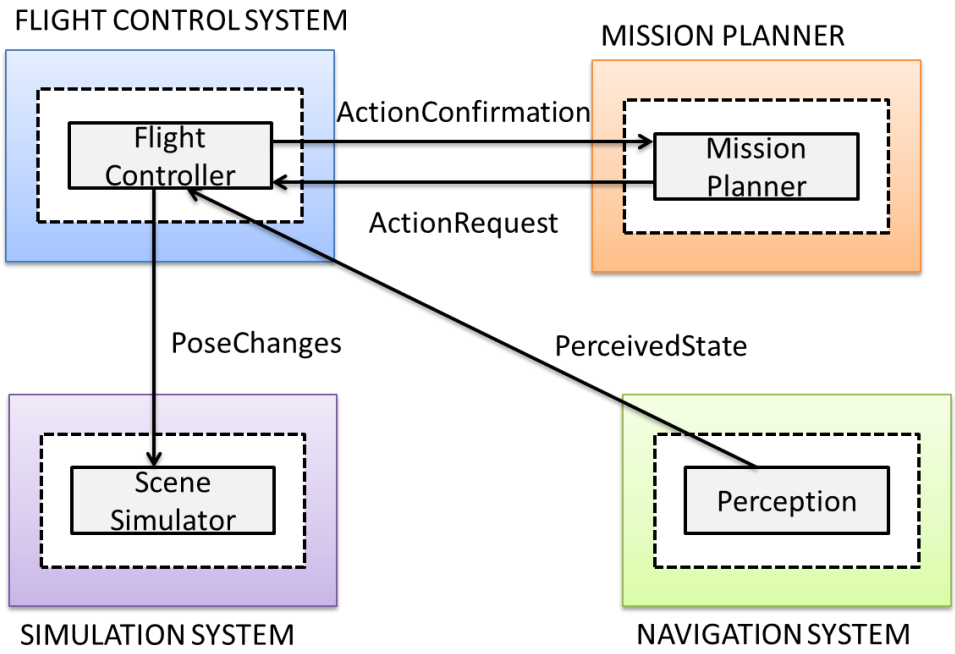


Figure 4-9: Flight Control System Communications

4.1.3.6. Simulation System

As its own name says, the simulation system is in charge to simulate the whole scene occurring in the arena during one complete contest try. A contest try takes a maximum of ten minutes as described in the competition rules (see Section 3.5). This module takes a master role. It advances the time of the simulation moving all the robots as they should move in the real competition.

Simulation system moves all the ground robots according to their defined behavior. Obstacles robots are moved, but also target ones, including their random component. Moreover, movements decided by the UAV are notified to the simulation system and updates the UAV position in the scene. The simulator is responsible to simulate all the interactions between the different agents too. The interaction between all different robots is simulated realistically following the competition rules.

Simulation system is composed by two components. The scene component is the main one and is in charge of realizing all the simulation. However, there is another component which is optional. Having the whole scene simulated, it is possible to show it to users translating it to 3D images. Then, a visualization component was created using Rviz, the ROS 3D tool. The simulation system is implemented using these two different components as separated nodes. All the simulation can work without the visualization system which is optional, but very helpful to see what is happening in the

scene. The communication between these modules is described as follows (see Figure 4-10):

- **PoseChanges**. Received from the Flight Controller. It carries the movement orders for the UAV.
- **SimulatedState**. Sent to the Perception System and optionally to the 3D Visualization System. It has the whole scene of the simulated world.

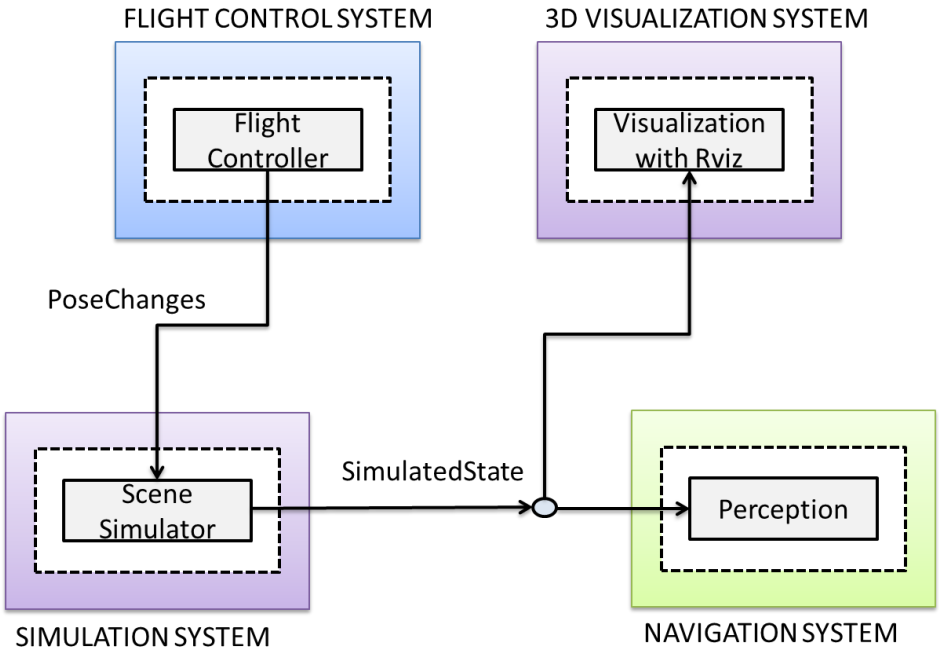


Figure 4-10: Simulation System Communications

5. IMPROVING THE SIMULATION PLATFORM

In the previous section was explained how the UPM Simulation Platform was designed. This section explains in detail the improvements realized in the Simulator. Specifically, the feature included in the new version is an optional mode for running the simulation faster, shorten the execution time.

To evaluate the different strategies, and, in general, any change in the system, it is critical to compute as fast as possible the entire simulation. As a single execution needs ten minutes, speed up simulations is a desired and highly important feature for our system. Additionally, in a non-deterministic system like our simulation system, where the robots have random behavior, do multiple executions is particularly important. The results obtained change in every execution, and then, to warranty consistence in results, enough executions should have completed.

This section explains how the speed up simulation feature has been implemented. The method used to gain time is based on serialize the processes and avoid the waiting time between cycles. For doing it, the new version is using a simulated time instead of the real time clock which was used to control the time. Then, as it is not depending on the clock, this method is asynchronous. To do this change, some synchronization methods have been applied that were not necessary in the previous real time version. In addition, the network configuration has been modified in order to minimize the overhead in the communication. Finally, a safety system has been designed to manage possible overhead similarly than the previous version. All of these changes are explained in detail in this section including the performance evaluation of the new system.

5.1. Time

The real time version of our system, updates the simulation time each cycle. The simulation time allows advancing in all the operations required to simulate the virtual environment. That includes not only simulate the different robots positions, but also update the decisions of the UAV which depend on time. Then, the simulation time must be visible to most of the processes, such as Scene Simulator or Mission Planner modules. To achieve it, the simulation time is updated only in the Scene Simulator, which takes the master role in the simulation system. Then, simulation time is included in the message sent by the Scene Simulator and it is propagated to all the modules. Thus, every module has the same simulation time in the same cycle to make its own calculations.

Knowing how works the simulation time, a modification was done in order to change the behavior of the system to a non-real time system. Real time systems have a fixed frequency for cycle execution, and consequently a fixed time between the starting of

two consecutive cycles. When a cycle finishes its computation, it waits until the start of the next one (see Figure 5-1). Removing waiting times is critical to make the simulation faster and take advantage of the available computation power. But starting next cycle just after finish the previous one is not enough. Simulation time has to be updated in a similar manner that it would do in the real time version. Then, it is possible to use a fixed cycle time equivalent to the real time version. As our system only uses this simulation time, to pretend that wait time have already passed can be implemented.

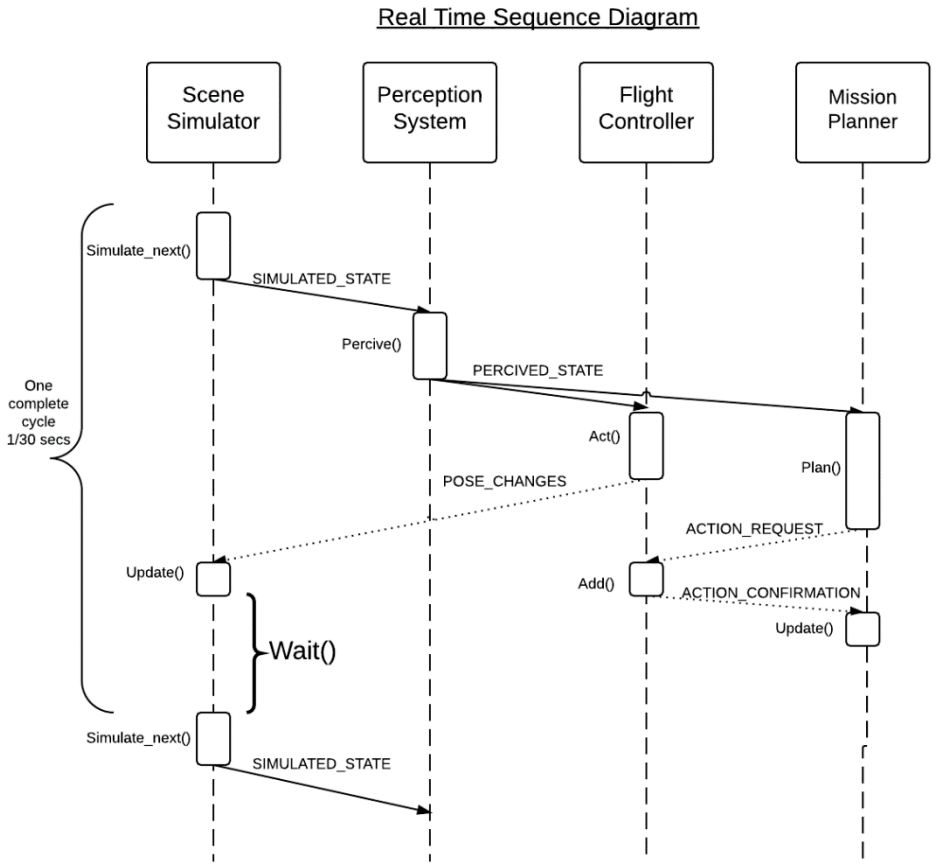


Figure 5-1: Real Time Sequence Diagram

In addition, is significant to consider the cycle duration. In the real time system, if one cycle is too short the computation could be longer than the cycle time and produce an overrun. If the cycle is too long, the refreshing time is longer and the precision of the simulation decreases. Achieve enough precision during the simulation is critical to obtain realistic behaviors of the agents.

For instance, in our case the real time system works at 30Hz. Using the frequency formula, we obtain a time of $0.0\hat{3}$ seconds per cycle. We decided simplify it to 0.03 seconds per cycle. It means that the cycle is $0.00\hat{3}$ shorter which is not a significant

fluctuate (10%). In any case the cycle is shorter, and then it is more precise, but more cycles are needed to complete the entire execution; the computation is slightly larger. Specifically in the IARC Simulator which simulates a ten minute-competition, with a 0.03 seconds cycle, it needs to execute 20000 cycles as is shown in Equation 5-1.

$$Frequency = \frac{1}{Period}$$

$$\frac{1\ second}{30\ cycles} = 0{,}0\hat{3}\ seconds\ per\ cycle$$

$$\frac{600\ secs}{0{,}03\ \frac{secs}{cyle}} = 20000\ cycles$$

Equation 5-1: Cycles Required for the Entire Simulation

However, simulation time modification is not enough to coordinate the system. In the real time version, on each cycle a new message is sent when the cycle time is over. The start of a new cycle does not depend on any other factor. The Scene Simulation module should know when the current cycle has ended in order to start immediately, but it cannot know. That is because how the architecture is designed.

We can see the whole Simulation System as a serial sequence of three processes: Scene Simulator, Perception System and a decision process consisting of the Flight Controller and Mission Planner which are executing in parallel. Figure 5-2 illustrates how the simulation data is propagated in each single cycle. First process is the Simulation Scene. It calculates all the changes in the position of the ground robots, the UAV and the time. The Simulation Scene sends a message to the Perception System and optionally also to a Scene Viewer to show the simulation with Rviz. The Perception System filters only the portion of the scene that the UAV can see and sends a message with that information to the Flight Controller and Mission Planner. Finally, Flight Controller and Mission Planner take the useful information and make its decisions. Then, only if necessary, the Flight Controller sends to the Scene Simulator the changes in the position of the UAV.

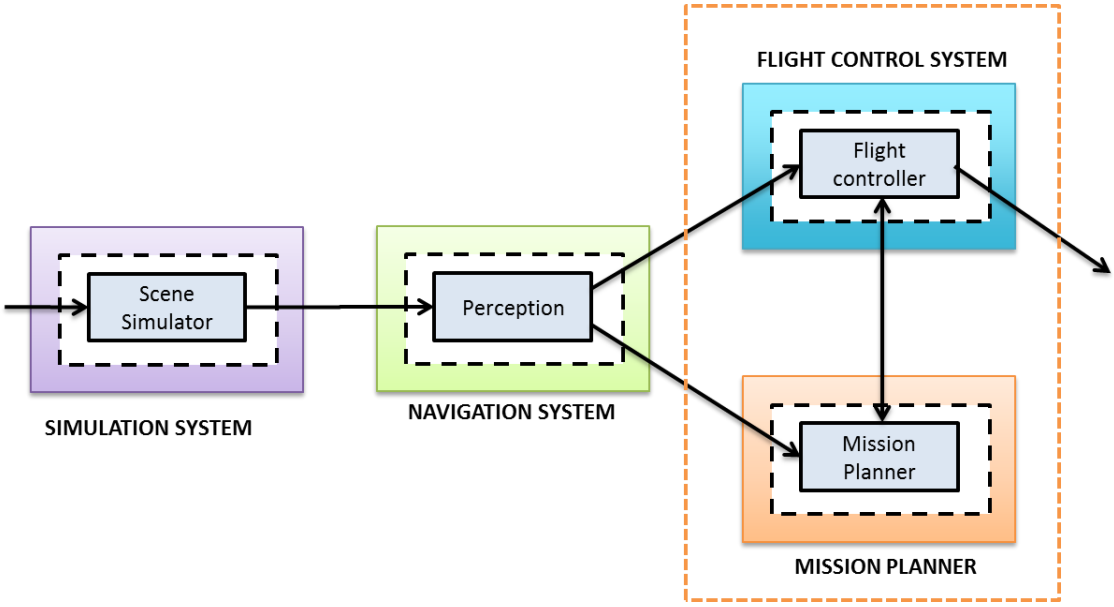


Figure 5-2: Individual Cycle Simulation Processing

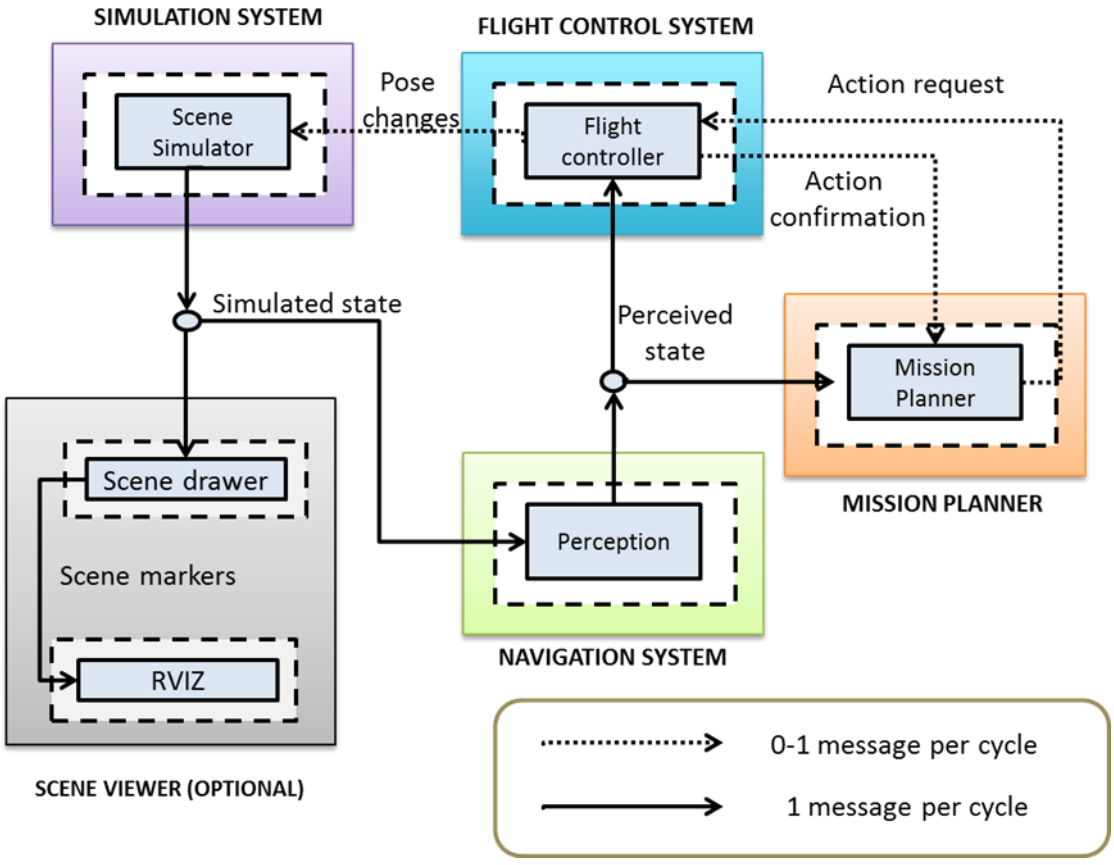


Figure 5-3: Real-Time Mode Architecture

First problem is due to the message sent to the Scene Simulator containing the changes in the UAV position. This message is optional in each cycle as is shown in the Figure 5-3. It is only sent if the decision is to move it. If not, the message is not sent and Simulation Scene cannot know if the process has finished. A change was done here in order to make this message not optional. This guarantees that the message is sent when the entire cycle is finished. The modification was to send a message without changes in the position of the UAV instead of not sending any message. Thanks to that, the Scene Simulation module can know when the entire cycle has finished and start the next one.

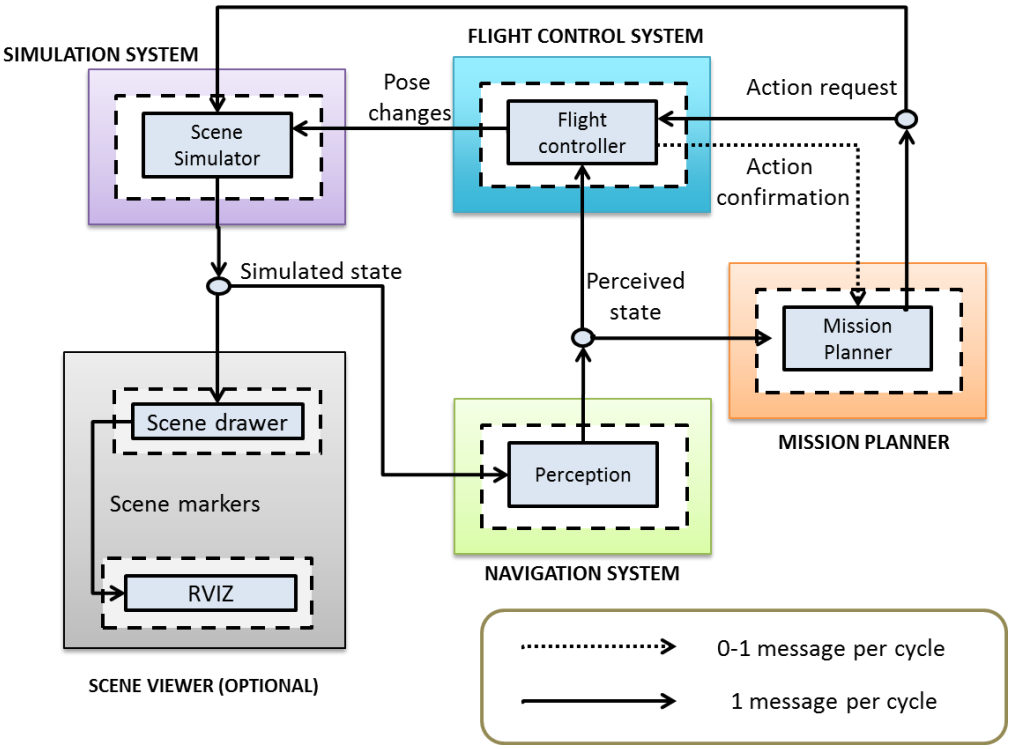


Figure 5-4: Speed-Up Mode Architecture

However, another modification is required because the Flight Controller and the Mission Planner are different modules executing at the same time and require synchronization for knowing when they have finished their executions. The condition to start next cycle in the Scene Simulator must be that both modules have finished their executions. To achieve this goal, the Scene Simulator needs to receive every single cycle the notification from the Flight Controller, but also another notification from the Mission Planner. Reproducing the same idea to reuse existing messages, the optional message *Action_Request* is used for Mission Planner notification

Action_Request is sent when the Mission Planner has decided the next action to be executed by the UAV. The message notifies that the work of the module has finished. The Scene Simulator module is subscribed to the *Action_Request* topic in order to receive the notification. As not every cycle is sent, it is possible to make it mandatory such as the *Pose_Changes* message. It was done using the type attribute of the message, creating another type that is only used to notify that the Mission Planner has finished its work without requesting any action. Thanks to this, the Flight Controller can ignore messages that not request any action. The new speed-up architecture obtained is shown in the Figure 5-4 and the Sequence Diagram in the Figure 5-5.

Additionally, there is a last optional message sent from the Flight Controller to the Mission Planner called *Action_Confirmation*. This message is only used to update the status of an action. It would be possible to create another message to notify that this update has ended. But it is not necessary because these updates have a fixed short execution time and if the network status is reliable, the update operation never causes a conflict with the next cycle execution. Then, it is better to avoid adding additional messages with more overhead.

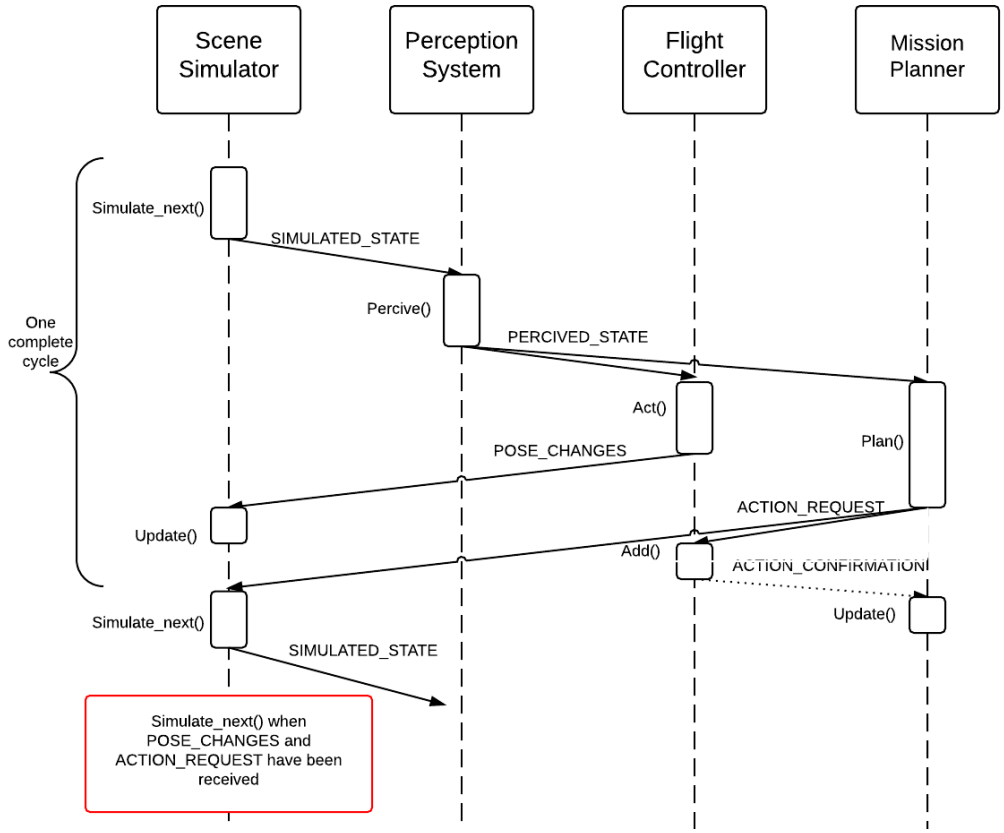


Figure 5-5: Speed-Up Mode Sequence Diagram

5.2. Network optimization

The modification described in this section is not necessary for the correct behavior of the system. But it eliminates a main restriction in the execution time and the speed up was low before this improvement was realized. This modification is addressed to minimize the overhead of the network.

In order to describe the improvement, it is convenient to start explaining how ROS connections work. ROS uses a network system based on the editor-subscriber paradigm. Users define topics, publishers and subscribers. Topics are the messages, and the users can define their contents. Publishers are the senders and subscribers the receivers.

The default configuration of ROS has a parameter called "*tcp_nodelay*". It is by default deactivated in the publisher and also in the subscriber class. *tcp_nodelay* provides to the user the option to use or not the Nagle's Algorithm on TCPROS connections. When *tcp_nodelay* is set to "*False*", the Nagle's Algorithm is used. Then, by default is active in the ROS connections.

Nagle's Algorithm is used in TCP connections in order to optimize the network when the "*small packet problem*" occurs. The "*small packet problem*" happens when very small packets are sent. Very small packets have big overhead because in TCP connection every packet needs a header of twenty bytes only in the transport layer. Therefore, Nagle's Algorithm waits during small amount of time and tries to join different small packages in order to send bigger, but less number of packages. Thus, the algorithm results in a more efficient use of the network.

However, this option affects directly to the speed of our system. It is due to its behavior, where each publisher sends a maximum of only one short message per cycle. For this reason, when the publisher sends a message, it has an overhead of waiting time for a new message which would add in the same package. Nevertheless, in our case this message never comes, and then, when a message is sent, it always has an overhead. But it never obtains the benefits of Nagle's Algorithm.

tcp_nodelay option do not affect the real time version of our system because in each cycle, every computation is executed at the expected time, even in the case that the option is disabled and produces an overhead as explained. It only affects the security margin of time to avoid an overrun, but the margin is enough wide to allow this overhead without problems. However, in our speed up version, the goal is minimizing the time per cycle. And this overhead increases notably the total execution time. Then, *tcp_nodelay* option has been turned on in the speed up version in order to gain time.

5.3. Speed-Up Real-Time Simulation System

Another important difference was detected between the new accelerated version and the real-time one. Overruns hardly ever happens. When an overrun occurs in the real-time version, the simulation can recover if the problem does not persist over time. A typical cause is that a processor is occupied by other process for longer than expected, but also network issues can produce unusual delays, among others. The real-time system is resilient to this type of delays which are common but not frequent and neither persistent. The clock of the simulation continuously advances and updates the scene. If an overrun occurs, the UAV modules compute the updated information from the simulation, recovering the normal execution state.

The problem of the speed-up mode is that it is based on a sequential processing. A complete cycle is processed from the beginning to the end without taking care about the time limit. When the overrun occurs, the processor continues working until finishing the cycle, no matter how large the delay is. This method is unrealistic; the simulation should manage the errors in the same way that the real-time version does.

To solve this inaccuracy problem, a timer was added to the system. The timer works to control possible overruns. If an overrun occurs, the timer starts "*real-time mode*". In this mode the next cycle starts similarly as real-time system does. The simulation scene is updated, sending the new information to the Perception System module. This "*real-time mode*" works until the system synchronization is recovered: when the delayed computations of previous cycles have already finished its computations. Thus, delayed messages of previous cycles are used to update the UAV position, but not as condition to start the next cycle.

5.4. Evaluation

After implementing the described optimizations, the new version was evaluated to measure the achieved speed up. A comparison with the old version was done. The total execution time of the old version is fixed. As it is a real time version, the execution must simulate the mission until the end. A simulation execution needs ten minutes if the mission does not end by other condition. This real time execution does not depend on the machine where it is executing, provided that the power is enough to warranty the correct behavior.

However, in the case of the speed up version, the total execution time depends on the machine which runs the simulator program. Just to take an idea of the time gained, an evaluation using our development environment was made. The development environment is an Ubuntu 13.04 Operating System running in VMware virtual machine. The hardware resources assigned are eight cores of a CPU model Intel Core i7-

3610QM, 2GB of RAM memory and 20GB of hard disk. We measured the time of the complete simulation session. Using twenty execution samples we obtained an average of 23.94 seconds per simulation. The minimum time of these executions was 19.71 seconds, and the maximum 31.67 seconds. The new version is 25.06 times faster in average under these conditions (see Equation 5-2).

$$Speed\ Up = \frac{Time_{old}}{Time_{new}}$$

$$Speed\ Up = \frac{600\ s}{23.94\ s} = 25.06$$

Equation 5-2: Speed Up of the New Simulator Version

It was possible to take under consideration another method to gain time: remove the networking overhead. ROS mechanisms for distributed computation based on modules are the main reasons why ROS was chosen for developing the project. In the IARC 7th Mission, the UAV has an important computation load, especially due to the complex Vision System. The rules of the tournament describe that additionally to the CPU mounted in the UAV; the teams can use an additional machine wireless connected to the UAV. ROS is a development environment that provides all the communication stuff between modules and at same time gives flexibility to allocate the modules in different machines. However, for only simulation is possible to make the executions in only one local computer and remove the network unnecessary overhead. Eliminate the network system is a possible optimization, but it was declined. Even all the overhead was removed; the benefit of the speed up is not enough to justify the change in the architecture. The gain would be few seconds per execution luckily because most part of the simulation time is computation. Additionally, a change of this magnitude makes the project less maintainable and flexible. It would remove the main advantage of ROS, which allows us to migrate easily the modules, just allocating them in the appropriate computer of the UAV. At same time, the simulator would be less realistic because the modules do not have to treat with possible problems from the network layer.

6. STRATEGIES DEVELOPMENT

The Mission Planner (explained in detail at Section 4.1.3.3) is the aerial vehicle software component in charge of taking the different decisions during the competition. Mission Planner role is highly important in this IARC competition. The time to guide the different ground robots is adjusted in order to be challenging. In addition, the different movement of ground robots is partially random which results in an unpredictable behavior of the different agents in the scenario. Thus, Mission Planner design is not a trivial task. On one hand, Mission Planner purpose is to maximize the number of robots guided. It must to be efficient. On the other hand, it also should consider unexpected situations which can occur with less frequency. Considering and resolving as better as possible these types of unexpected situations results in a reliable planner.

The decisions of the planner can be split into two different types: Selection Decisions and Searching Decisions. Selection Decisions are those related with guiding a robot or not. It includes which ground robot is going to be guided when the UAV see several of them and when a robot is discarded. Otherwise, Searching Decisions are functions to search ground robots through the arena.

It is to note that strategies development is iterative process. After developing and evaluating one strategy, it is possible to detect repeatable facts, acquiring knowledge about the scenario. This knowledge can be used to develop new strategies. For instance, if we know that guiding a robot from a specific area requires at least one minute, search robots in this area in the last minute of the mission should be discarded.

6.1. Mission Time

The time is one of the most important factors of the mission. A minimum of seven ground robots has to be guided to complete the mission. To complete this task in ten minutes is challenging, but to guide the ten robots seems to be near impossible. Losing as less time as possible during the mission is critical for obtaining good performance in the competition. To measure the importance of time, an experiment was designed. This experiment measures the performance when the aerial robot is stopped for one minute during the competition. Three scenarios have been compared: one where the UAV flights normally, other where the UAV is stopped during the first minute of the mission, and the last where the UAV is stopped during the last minute.

This experiment has two purposes. First, it tries to understand in which degree losing time can affect to the results of the mission. It is important to check the influence of this factor. If the UAV has enough time, it will change the priorities. Some strategies can be

influenced significantly by this factor. For instance, to decide spending time or not saving robots which are going to abandon the arena.

The other reason for doing this experiment is to try to see if the influence of the time changes during the mission. This is why we compare the effect of losing the last and first minute of the mission. At the beginning of the mission, the scenario is different than at the end. At the beginning, all the ground robots are in the arena. However, during the mission, ground robots abandon the arena either because they have been guided or moved by themselves out of the limits.

6.2. Searching Strategies

One of the main decisions that the UAV have to take is where to find a candidate ground robot to be guided to the goal. Searching function selects the point where the UAV goes looking for new ground robots. The UAV is continuously looking for candidate robots to guide within its vision range, until it decides to select one of them. For searching, there are infinite possible strategies from go to a random or fixed point, to use a complex decision using all the information previously known by the UAV. For instance, the memory of the robot can store information such as when and where it has seen the different ground robots, or the time left to finish the competition. Following, the different implemented strategies are explained.

6.2.1. Center Point Strategy

Center point strategy is very simple, the UAV always go to the center point of the arena. There, it waits until it decides to guide a robot in the vision range.

6.2.2. Random Strategy

In this strategy, the UAV selects a random point in the field and goes there. When it arrives the random point, it selects a new random point to go. And so on until the UAV decides to guide a ground robot.

6.2.3. Non Visited Strategy

Non Visited Strategy tries to improve the Random Strategy. The idea behind this strategy is to discard those areas where the expectation to find a robot are lower. Areas already explored recently by the UAV without seen any robot seems to be good candidates for discarding. The Non Visited Strategy uses the list of points visited by the UAV in the last minute to generate a discarding area. The strategy generates a random point in the same way it does in Random Strategy, with the difference that if the point to search is in the discarding area a new searching point is generated. This process is

repeated a maximum of three times, the fourth point generated cannot be discarded even if it is in the searching area. Thus, this strategy prioritizes not going to already visited areas.

6.2.4. Square Route Strategy

Square Route Strategy is a route based strategy. In this kind of strategy the UAV navigates through different points of the map following a route. In the case of the square route it goes through four points distributed as a square in the arena (see Figure 6-1). It tries to cover under the vision of the UAV most of the full arena.

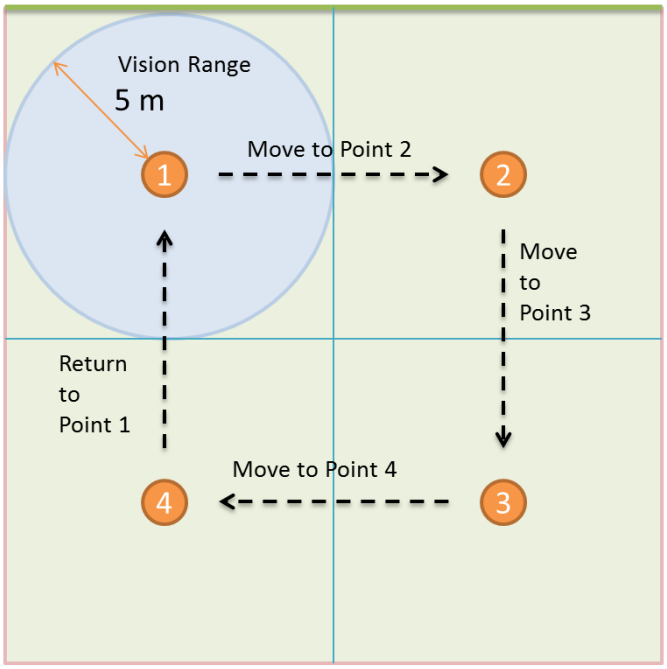


Figure 6-1: Square Route Strategy

6.2.5. Diamond Strategy

Diamond Strategy is similar to Square Route Strategy, but using different positions for the route points. They have a diamond distribution moving through four different points (see Figure 6-2). This route covers less area than the square route because the robot cannot see the corners of the arena.

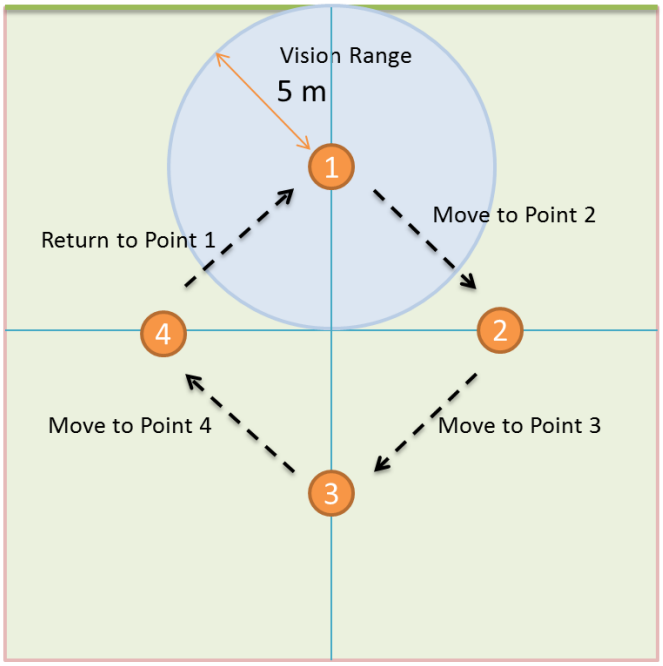


Figure 6-2: Diamond Route Strategy

6.2.6. More Robots Strategy

The idea behind the More Robots Strategy is to use the memory of the UAV in order to increase the possibilities to find a robot in a specific area. Intuitively, good candidate area to search a robot may be one where robots were previously seen. This seems more likely if robots have been seen short time ago. If they have been seen long time ago, the possibility that they have moved to other area seems to be higher. In addition, in the areas where more robots have been seen, it seems more likely that some still be on the area.

The More Robots Strategy needs to use the long term memory of the aerial robot. Specifically, the strategy uses a memory with the last position of the ground robots stored by the UAV. Using this information, the strategy can select the area where more robots have been seen in the last minute. However, if the UAV does not find any robot in that area, another strategy needs to be used. It is then, a combined strategy.

6.2.7. Last Robot Seen Strategy

Last Robot Seen Strategy tries to find the last robot seen by the aerial robot vehicle. The UAV goes to the last known position of the last seen ground robot. If a ground robot was seeing long time ago is more difficult to find it in the same area because it has been moving during all the time elapsed. Thus, the UAV go to the last robot position only if it was seeing in the last minute. This is a combined strategy. In the case that after

arriving to the position the UAV cannot see any ground robot, another strategy should be selected.

6.2.8. First Near Strategy

This strategy is based on the hypothesis that is better to guide the ground robots located near the goal line first. It seems better than guide those which are far from the goal. There are two ideas that may support this hypothesis. First, it seems that guide a robot near the goal requires less effort. Less distance to the goal requires, in most of the cases, less interaction and time for guiding a robot. Thus, searching first the easier robots seems to be a good idea. The second idea for supporting the hypothesis is related with the evolution of the environment. When the UAV guides a robot located far from the goal, there are many probabilities that it find an obstacle. An obstacle can be a specific "*obstacle*" or another "*target*" ground robot which is located closer to the goal. For this reason, if the UAV guides the near robots first, it seems that the scenario will become easier for guiding the further ones later.

The implementation of First Near Strategy has been designed in order to check this hypothesis. For this reason, another strategy has been created: First Far Strategy. First Far Strategy is the opposite strategy to first near; it prioritizes far robots. The comparison of results from both strategies can be used for checking the hypothesis.

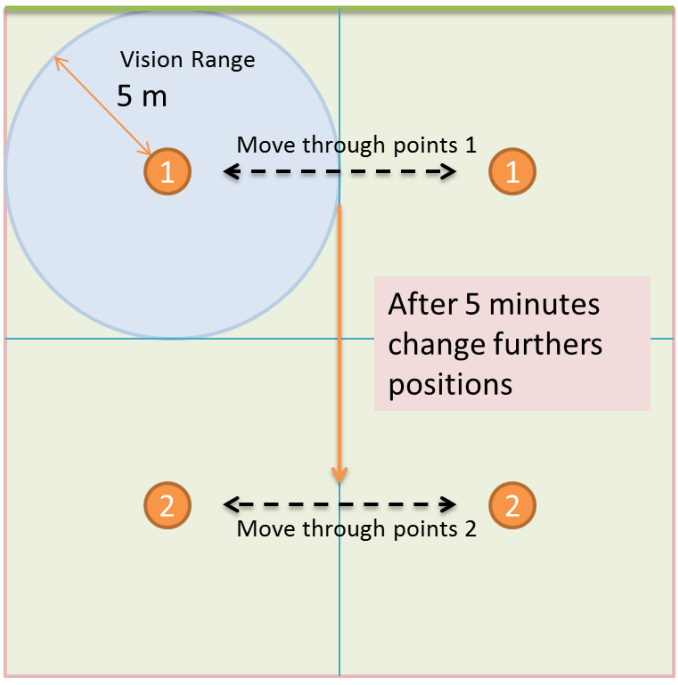


Figure 6-3: First Near Strategy

First Near Strategy implementation split the arena into two areas: the area near the goal and the area far from the goal. The UAV is searching the first five minutes in the area near the goal and the last five minutes in the other area. Specifically, the UAV covers most of each areas moving between two points located in the extremes of each area (see Figure 6-3).

6.2.9. First Far Strategy

First Far Strategy is the opposite strategy to First Near Strategy. It prioritizes searching the ground robots far to the goal. The UAV searches the first five minutes in the area far to the goal and the last five minutes in the area close to the goal, just the opposite than the First Near Strategy (see Figure 6-4).

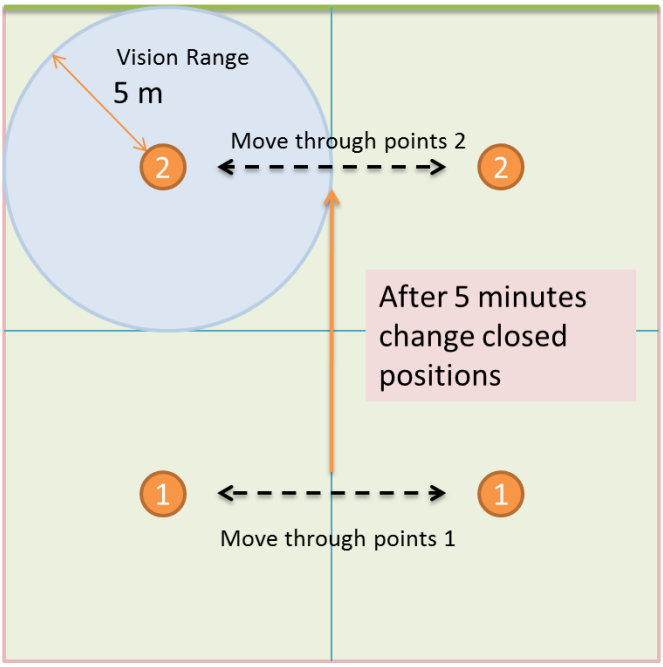


Figure 6-4: First Far Strategy

6.2.10. Route Strategy Alternative

Route Searching Strategy is executed several times during a single mission. The UAV looks for a new robot each time it does not see any candidate robot. The route strategies have several points to visit. We have proposed as base case that the UAV memorizes the last point visited to continue the route when searching function is execution again.

However, it is interesting to evaluate if starting again the route from the first point may have an impact on the efficiency. This alternative prioritizes the first points of the route which are visited more frequently and before than the others.

6.2.11. Combined Strategies

Strategies like More Robots Strategy or Last Seen Robot Strategy are combined searching strategies because they need a secondary strategy to execute after the first one fails. In addition to these strategies, it can be useful to change the main strategy during the mission. It can be useful because the scenario is changing during the competition. Some strategies can adapt better than others to some scenarios and vice versa.

Combined strategies can be very useful but are hard to evaluate and optimize. There are many possibilities to combine the different strategies and they are combined by a change condition which needs to be defined. Change condition defines the situation where the strategy must be changed. Any information can be used to take this decision. The goal of the change condition is to detect that the current scenario fits better with other strategy. Examples of useful information to take this decision is the time elapsed since the start of the mission, the number of robots already guided or time elapsed without find any robot.

6.3. Selection Strategies

An important function of the UAV's planner is to select the most appropriate ground robot to be guided. Additionally, another important responsibility is to detect when a robot is under a situation in which is more advisable to discard it. Selection Strategies are the different strategies for deciding about robot selection and discarding.

Selection Strategies can be split into two components: a Choice Strategy and Discarding Functions. Choice Strategy decides which robot to guide among the several ones in the vision range. Discarding Functions decides under what conditions a ground robot should be discarded.

A Selection Strategy is defined by the combination of its choice strategy and discarding functions used. Different discarding functions can be used simultaneously, applied under different conditions. Also is possible to not use any discarding function.

6.3.1. First Near Choice

The First Near Choice is the most basic of choice functions. The UAV selects the ground robot more near to the goal line inside its range of vision (see Figure 6-5). If the aerial vehicle only can see one ground robot, it will select this one to be guided. In the case it cannot see any possible target, it will continue searching using the searching strategy set on its configuration until a ground robot is located. Once a ground robot is selected, the UAV will guide it to the goal.

6.3.2. First Far Choice

The First Far Choice is the inverse strategy to First Near Choice. It works similarly, but the UAV selects the ground robot farthest to the goal line when there are several options instead of the nearest (see Figure 6-5). [9]

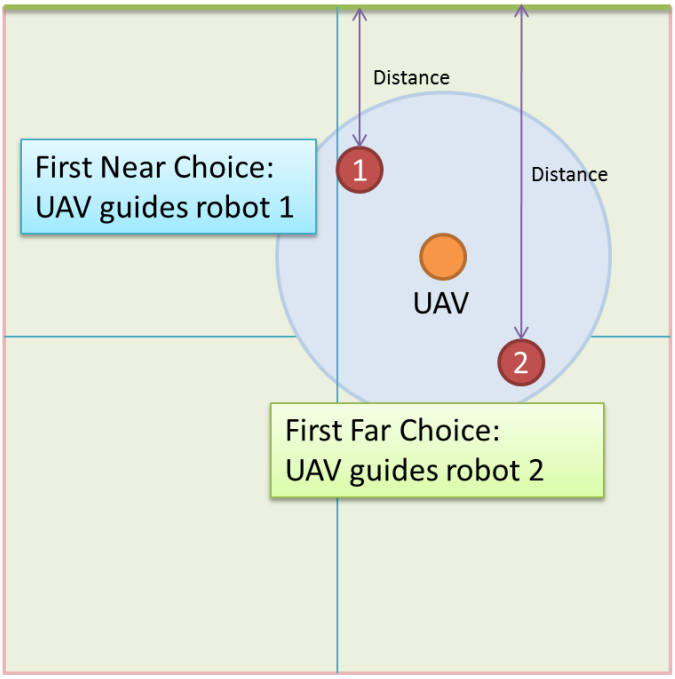


Figure 6-5: Selection Choices

6.3.3. Discarding Guided Robots

In the basic case, the UAV guides the robots until they have already abandoned the arena. To gain time a good strategy can be to discard a guided robot when it is already directed to cross goal by itself. The strategy determines if the UAV should abandon the selected robot calculating the likely trajectory of the ground robot. The current trajectory considered is the path walking before the ground robot turns round. The UAV abandon the ground robot when it thinks that the ground robot is going to cross the goal line by itself and do no need more interaction. However, due to the random behavior of the robots and the errors of accuracy in the non-simulated scenario it is difficult to calculate precisely when is appropriate to abandon a ground robot or not. Our system calculates the trajectory of the ground robot without consider the random change in its direction, but adding a security margin of 0.5 meters. This security margin is more useful than more sophisticated predictions because it can be adjusted easily. With this strategy, there are three possible predicted situations:

1. **Need more interaction.** The trajectory prediction for the ground robot determines that the robot is not going to cross the goal line and more interaction is required. The prediction is correct in almost all the cases, but hardly ever a robot may cross the line resulting in a "*waiting unnecessarily*" case.
2. **Do not need more interaction**. The trajectory predicts that the ground robot does not need more interaction. Then the UAV can abandon the ground robot because the robot is already going to the goal by itself. However, if the prediction fails, the robot will keep in the arena resulting in a "*not guided to the goal*" case.
3. **Probably not need more interaction**. The predictions suggest that the ground robot is going to be in the margin area. The UAV knows that the ground robot probably cross the line, but the accuracy of the prediction is not enough to determine it.

As the situations shows, there are two different cases to minimize for optimizing the problem. "*Waiting unnecessarily*" happens when the UAV waits, but the robot achieves its goal without help. Then, the aerial robot wastes time watching how it goes to the goal instead to guide other robots. The other situation is "*not guided to the goal*". It happens when a ground robot do not achieve its goal after being abandoned. The consequences of this situation are more difficult to predict. The ground robot keeps in the arena, but normally near to the goal and it is not rare that after rotate few times, the robot finally cross the line by itself. However, this robot also can abandon the arena for other sides of the square or be an obstacle to other robots. For these reason, this strategy tries to avoid "*not guided to the goal*" cases and minimize "*waiting unnecessarily*" ones. For this reason, the UAV waits when it is predicted the "*probably not need more interaction*" case.

6.3.4. Discard Near Obstacles

This discarding strategy is used to avoid difficult guiding situations. When a ground robot is too close to other robots or other obstacles, it is discarded as selection candidate. To guide a robot near others normally requires extra interaction in order to avoid them as obstacles and also any kind of unexpected situations. This strategy tries to avoid these situations discarding the robots which are separated from others less than 0.6 meters taking the center of their positions as reference.

7. EVALUATION

After developing a strategy it is necessary to evaluate it in order to quantify its performance and understand its behavior. This section explains the experiments realized and their results.

7.1. Evaluation Method

The different strategies or any other experiment proposed have to be executed in order to compare their results. Comparing different behaviors in a non-deterministic scenario requires a significant number of executions. Each execution is unique and cannot be consider representative of the whole possible results, just a single sample. For this reason, repeating the executions a significant number of times results in a reliable prediction.

The execution time has been reduced drastically thanks to the changes done in the Simulator Architecture (see Section 5). However, the time for a single simulation execution still requires tens of seconds. With tens of seconds per execution, obtain a significant number of samples results in an expensive time cost. Executions of hours to obtain reliable results always slow down or difficult the development process. This case is not an exception; it is highly desired to obtain the results of an execution set as faster as possible.

With these issues in mind, we decided to balance the number of executions with the time cost. We decide to execute 200 times each Mission Planner configuration to be evaluated. It takes a considerable amount of time, but results are enough reliable to extract useful knowledge from the experiments. With this number of executions, results still appreciate little variations executing the same configuration. We have considered for our experiments that these little variations do not affect enough our current experiments. However, if a specific experiment requires extreme accuracy, it is possible to increase the number of executions.

The different strategies are evaluated independently, but into a complete simulation session. The final goal is to optimize the global performance of the Mission Planner, not the particular results obtained by a function. To illustrate this, we can consider the Searching function. It seems that the Searching function goal is to find a robot as quick as possible. However, if the UAV finds a robot which is far from the goal or near an obstacle, it could be more complicate to guide it than other which requires a little less time to find. One searching strategy may be the fastest one for searching robots, but not the most efficient to complete the mission.

The manner decided to execute our experiments is to maintain the same base configuration changing only one of the strategies type or parameter. It is important to take under consideration that each configuration of the Mission Planner can affect in different way the global strategy. A searching strategy could combine well with a selecting strategy, but could be inefficient with other. Nevertheless, using a simple base configuration, the different strategies do not show significant differences.

7.2. Time Influence

An experiment was proposed in order to analyze the impact of the time through the mission. The experiment compares shorter flight times with the same Mission Planner configuration. Specifically, in one experiment the aerial robot starts the competition when all the agents have been moving for one minute. In the other experiment, the aerial robot finishes it flight one minute before the competition ends. In both cases, the aerial robot flight one minute less than the competition duration. However the minute of losing time is located in different time slots in order to ascertain if time factor change during the mission.

	Robots Guided	Robots Out	Time for Guiding 7	Mission Completion Rate
Ten Minutes Flight	7.9	0.625	456.295 s	93%
Nine Minutes Flight starting at minute one	7.14	0.78	526.49 s	72%
Nine Minutes Flight finishing at minute nine	7.24	0.64	462.97 s	79%

Table 7-1: Shorting Flight Time Average Results

Table 7-1 shows the average results of 100 executions of the two experiments proposed and their comparison with the same Mission Planner configuration under regular flight time of ten minutes. The results of the table conclude that losing one minute has key influence in the performance of the mission. The Mission Completion Rate of the configuration evaluated (93%) decreases to 79% in the case of losing the last minute and 72% when is the first one. This is because the time is adjusted to represent a challenge. To take better picture, the average for guiding seven robots and then complete the mission is more than seven and a half minutes. The margin is in average less than two and half minutes to complete the minimum requirement. And this is using a configuration with high Mission Completion Rate, other configurations or changes in the robot can consume part of this margin. Moreover, same conclusions can be inferred for the average of robots guided. This average also changes significantly from 7.9 to 7.14 and 7.24, respectively.

These results conclude that the mission time is adjusted to represent a challenge. It is critical for aerial robots in the competition to optimize the time for obtaining the best performance in the mission. However, the results also show interesting differences between losing time at the beginning or at the end of the mission. When the mission starts after one minute, the performance is worse in all the measures taken. Fewer robots are guided, more robots are disqualified abandoning the arena and the mission completion rate is significantly lower. Thus, guiding ground robots quickly is more important for the success of the mission. The scenario gets easier and the risk of ground robots disqualification decreases.

7.3. Selection Strategies

	Robots Guided	Robots Out	Time for Guiding 7	Mission Completion Rate
First Near	7.9	0.625	456.295 s	93%
First Near + Discard Guided	7.81	0.535	451.315 s	91.5%
First Near + Discard Near Obstacles	7,88	0,665	456,96 s	91%
First Near + Discard Guided + Near Obstacles	7.84	0.53	446.195 s	90%
First Far	7.625	0.725	469.33 s	82.5%

Table 7-2: Selection Strategies Averages of 200 executions

Results obtained after 200 executions are shown in the Table 7-2. The comparison between the two *Choice Strategies* shows that First Near Choice Strategy is substantially better than *First Far Choice Strategy*. All the average statistics are better for the First Near case. It guides in average more robots and less of them abandon the arena. Also the Completion Mission Time is lower. But the most significant difference is the Completion Mission Rate. When the First Near Choice is working, the rate is 93% of successful mission against the 82.5% when is First Far Choice. This confirms the hypothesis that First Far Choice selects a robot which requires more effort for being guided to the goal.

With regard to Discarding Functions, three configurations have been evaluated. One with Discard Guided function enabled, other with Discard Near Obstacles function enabled and the last with both. The results of these functions show a slightly worse performance than the Base Strategy, First Near without discarding functions enable. The Mission Completion Rate decreases between 1.5 and 3 percentage points. The average of ground robots guided shows a slightly decrease, around 1% of difference. However, when Discard Guided function is enabled, the robots disqualified decreases more significantly up to 15%, from 0.625 Robots disqualified in the Base Strategy to the 0.53 when both functions are enabled. When both functions are enabled the average time is

also reduced in up to 10 seconds. Thus, the functions gain time, especially Discard Guided function, but reduce slightly the number of robots guided which is the goal of the mission.

Talking about the distribution of the data, we can see in Table 7-3 that the standard deviations of the robots guided are between 0.97 and 1.278, not especially significant. We can see in Figure 7-1 that this deviation is not large; most of samples are grouped in the same area, with very few atypical cases (10 robots in the upper limit or 5 or less robots in the down limit).

	First Near	First Near + Discarded Guided	First Near + Discard Near Obstacles	First Near + Discard Guided + Discard Near Obstacles	First Far
Average	7.90	7.81	7.88	7.84	7.625
Variance	1.075	0.979	0.940	1.482	1.633
Standard Deviation	1.037	0.989	0.970	1.217	1.278

Table 7-3: Selection Strategies statistics of robots guided with 200 executions

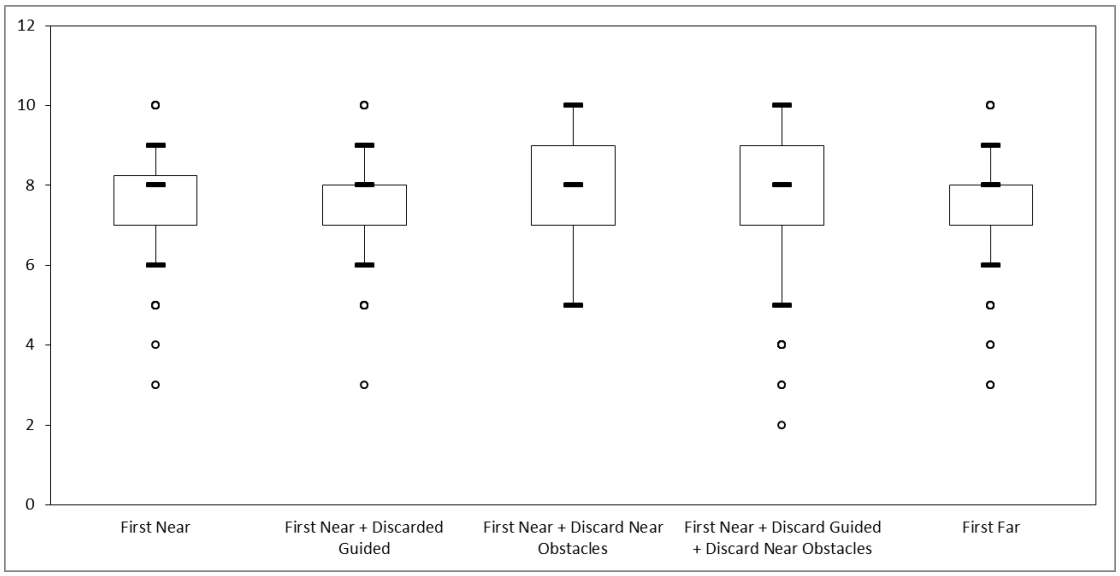


Figure 7-1: Selection Strategies distribution of robots guided with 200 executions

7.4. Searching Strategies

	Robots Guided	Robots Out	Time for Guiding 7	Mission Completion Rate
Center Point	7.685	0.58	465.335 s	89%
Random	7.625	0.77	481.875 s	86%
Non Visited	7.44	0.715	488.485 s	84%

Diamond	7.9	0.625	456.295 s	93%
Square Route	7.84	0.63	467.055 s	92%
Square Route + Starting Again	7.59	0.65	464.845 s	90.5%
First Near	7.665	0.69	477.175 s	89%
First Far	6.82	0.805	494.505 s	66%
Last Seen + Center	7.76	0.59	455.115 s	91%
Last Seen + Diamond	7.59	0.74	475.15 s	84%
More Robots + Center	7.66	0.66	458.41 s	89%
More Robots + Diamond	7.825	0.725	458.02 s	88.5%
Diamond → Random	7.695	0.685	466.345 s	90%
Random → Diamond	7.69	0.76	473.935 s	89%

Table 7-4: Searching Strategies Averages of 200 executions

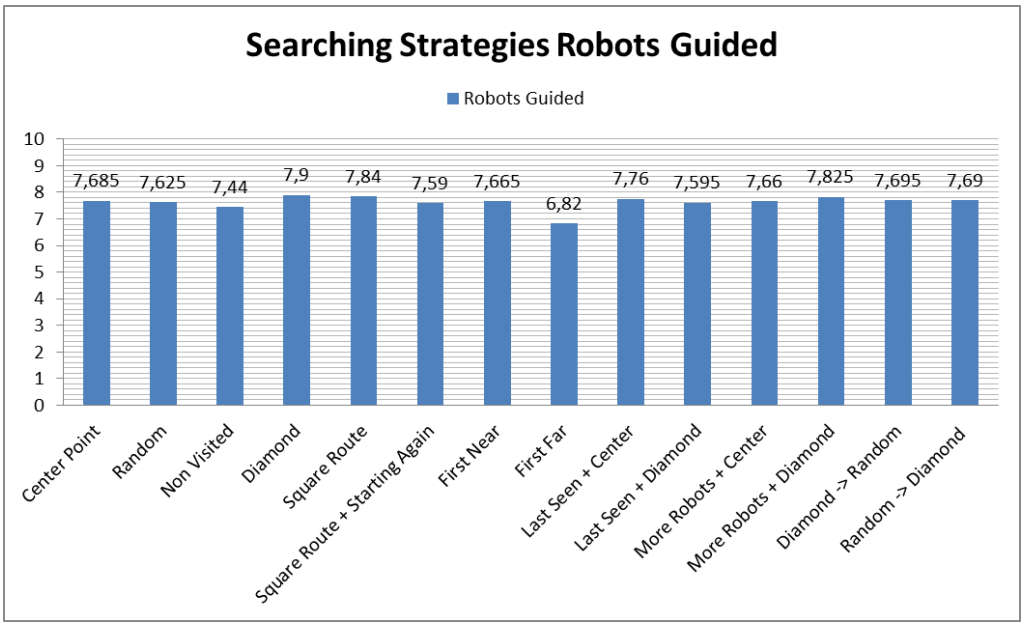


Figure 7-2: Robots Guided by the different Searching Strategies

Figure 7-3: Mission Completion Rate (when at least 7 robots are guided) of the different Searching Strategies

7.4.1. Center Point

Center Point Searching Strategy performance is intermediate-high compared to the others strategies evaluated. It has 89% of Mission Completion Rate and an average of 7.685 robots guided to the goal. The performance is worse than the best strategy with a difference of 4 percentage points of Mission Completion Rate and more than 0.2 robots guided less in average. However, the performance is still being quite good, superior to other strategies proposed like Random Strategy. Additionally, the Center Point Strategy shows the best performance avoiding robots disqualifications with only 0.58 robots disqualified in average.

Even this strategy is simple and its vision does not cover the entire arena surface; this strategy demonstrates the importance of the central area. All robots start from the central area and they periodically turns 180 degrees. This type of movement covers a significant length of terrain, making easy that robots are visible to the UAV when they approximate to the central area. Each robot movement is partially random, which result in difficulty to predict the exact movements, but the central area is where all robots converge. For this reason, any robot can move to the central area, while it is difficult for a robot to cross the arena from one side to another.

7.4.2. Random and Non Visited Strategies

With 7.625 robots guided in average and 86% of Mission Completion Rate, Random Point Searching Strategy shows an intermediate-low performance. The Random Strategy moves randomly through the arena surface searching for the robots. The strategy needs less time than others to find a robot, but results in worse performance due that it does not find the best robots for guiding.

Additionally to the Random Point Strategy, another random searching strategy was evaluated. Non Visited Strategy uses the same base behavior, but it discards the areas already visited if the UAV has not seen any ground robot in the last minute. However, this strategy decreases a little the performance. It obtained 7.44 robots guided in average and 84% of Mission Completion Rate.

7.4.3. Route Strategies

Route Strategies achieved the best performance for Searching Strategies. Moreover, Diamond Strategy obtained better performance than Square Route Strategy which is the second best of the strategies evaluated. Specifically, they obtained 93% and 92%, respectively, in the Mission Completion Rate and, 7.9 and 7.84 in the average of robots guided to the goal. The success of these strategies is because they cover most of the arena surface with the UAV vision; the UAV can find easily a good robot candidate to guide.

In addition to normal route strategies where UAV remembers which is the next point to go after guiding a ground robot, an alternative route strategy was evaluated. We enabled the *"start again*" option in the Square Route Strategy. With start again option, the UAV goes to the first point of the route strategy each time it returns to the *Find* state from other state. This can happen when the aerial vehicle guides a ground robot to the goal, for example. With this option enabled, the last points of the route need more time to be visited and the UAV is more focused in the first points of the route. For the case of Square Route, it decreases slightly the performance with 7.59 robots guided in average and 90.5% of Mission Completion Rate. It is a difference of 0.25 robots guided less and 1.5 percentage points with the base Square Route Strategy.

7.4.4. First Far and First Near

The experiment confirms that First Near Searching Strategy is better than First Far. Even there are non-refinement strategies; the results obtained by the First Near Searching Strategy shows an intermediate-high performance. The First Near Strategy gets 89% of Mission Completion Rate and 7.665 robots guided to the goal in average. On the other hand, the First Far Strategy shows a low performance with 66% of Mission

Completion Rate and 6.82 robots guided. This is 34.85% of improvement in the Mission Completion Rate and 12.39% in the ground robots guided.

Thanks to this experiment, we can infer that searching first near the robots located in the goal line is an efficient strategy. While starting searching far to the goal is unproductive.

7.4.5. Combined Strategies

Our experiments included some combined strategies. Specifically, we have evaluated six different strategies. Last Seen Strategy and More Robots Strategy were evaluated with two different auxiliary strategies: Center Point and Diamond. And the last strategies evaluated are pure combined ones: Diamond then Random Strategy and the inverse, Random then Diamond Strategy.

Last Seen and More Robots strategies are based on the same idea. They try to find a ground robot already seen. The UAV goes to the point or area where it considers more possible to find any of the ground robots seen before. These strategies show little different performance, but in general the performance is intermediate-high, quite similar to the Center Point.

Last Seen combined with the Center Point improves the performance of the Center Point Strategy. It gets 91% of Mission Completion Rate and 7.76 robots guided. It also gets the best average time to complete the mission with 455.115 seconds, even better than Diamond Strategy. However, when the strategy is combined with Diamond, the performance decreases substantially. Mission Completion Rate is 84% and 7.595 robots are guided.

More Robots shows less different results between their two auxiliary options. When it is combined with Center Point, it gets a slightly better Mission Completion Rate: 89% against 88.5% with the Diamond Strategy. However, with the Diamond Strategy the average of robots guided is higher with 7.825 against the 7.66 guided when the Center Point Strategy is used. Also it is notorious that Center Point option performs better avoiding disqualification of robots in both cases.

The combination of Diamond and Random Strategy is evaluated by executing first one or the other. The UAV executes one strategy until it has guided three ground robots, then, it executes the other strategy. This experiment tries to find if any of these strategies performs better under a specific scenario with many robots or with few ones. The results show an intermediate performance between the both strategies combined. When Diamond is executed first, 90% of Mission Completion Rate and 7.695 robots guided, the performance is not higher than the Diamond Strategy, but neither as low as the Random Strategy. When Random strategy is executed first, the results show similar

performance with 7.69 robots guided and 89% of Mission Completion. At first look, the strategies do not show significant difference between performances at first part or the mission or once the mission is in advanced state. However, there are many factors which influence the mission. Then, taking more detail information for deeper analysis should provide us valuable knowledge.

7.4.6. Distribution of Data

The distribution of the data is not very different between all the strategies. Table 7-5 shows the standard deviation of robots guided of each strategy. We see that the standard deviation does not change too much between the different strategies, keeping around 1.1. In addition, we can see in Figure 7-4 that the distribution is quite similar with the exception of First Far strategy which is significantly worse than the others.

	Average	Variance	Standard Deviation
Center Point	7.685	1.403	1.184
Random	7.625	1.311	1.184
Non Visited	7.44	1.303	1.141
Diamond	7.9	1.075	1.037
Square Route	7.84	1.190	1.091
Square Route + Starting Again	7.595	1.850	1.360
First Near	7.665	1.149	1.072
First Far	6.82	1.977	1.406
Last Seen + Center	7.76	1.128	1.062
Last Seen + Diamond	7.59	0.997	0.998
More Robots + Center	7.66	1.602	1.266
More Robots + Diamond	7.825	1.462	1.209
Diamond -> Random	7.695	1.258	1.122
Random -> Diamond	7.69	1.381	1.175

Table 7-5: Searching Strategies statistics of robots guided with 200 executions

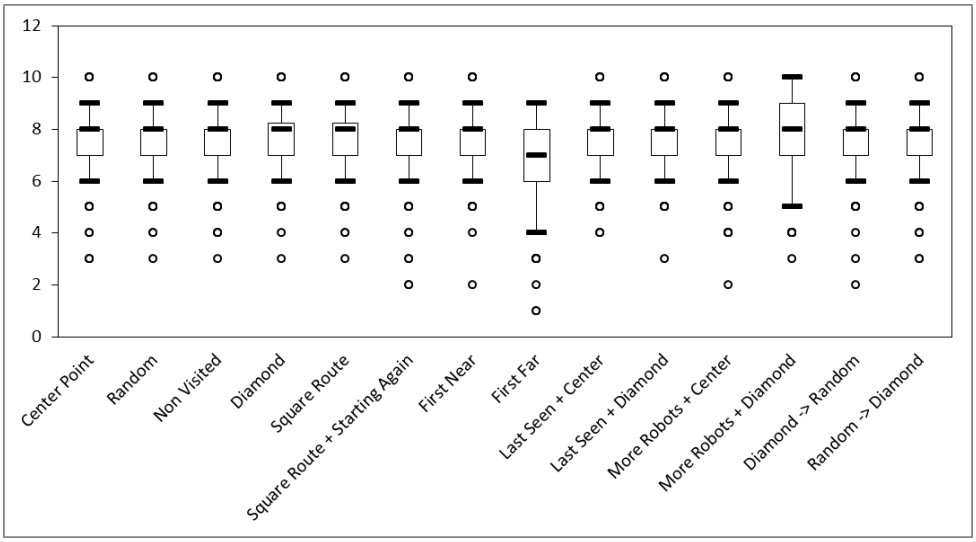


Figure 7-4: Searching Strategies distribution of robots guided with 200 executions

8. CONCLUSIONS AND FUTURE WORK

This Master Degree Project has been developed to be a contribution to the UPM team for the 7th Mission of the International Aerial Robotics Competition. This competition represents a significant challenge in the research on aerial autonomous robots. The mission proposes a highly dynamic scenario where the aerial vehicle developed must guide several ground robots to a particular goal area. The method to guide the ground robots is interacting physically with them to change their behavior. In addition to this challenge, all the robots move in partially random way and there are moving obstacles which cannot be touched.

The project has been focused to the improvement of the Mission Planner, the software component in charge of taking decisions autonomously during the competition. Specifically, different strategies for the aerial robot have been studied. It has resulted in a planner with a higher performance as well as the acquisition of valuable knowledge about the mission which can be used to define new strategies. Moreover, to study the strategies, a Simulation Platform has been used. The Simulation Platform was already developed by the UPM team, but a new feature was included as part of this work: speeding up the simulation.

The project has consisted of four parts. The first part of the work has been a review of the current status of the topics involved in the project. The review has covered the most relevant material for developing this project. It includes many different topics, from those more general in the area of robotics, to the more specific ones used only for this project. In the area of robotics, a general overview has been done for understanding the most important features of robotics field. In addition, the different programming paradigms have been reviewed in order to know the evolution of robotics and the issues related with its programming. Going deeper, the project is focused on aerial autonomous robots, and then aerial robots and autonomous robots have been studied. Finally, it has been necessary to know in detail the rules of the competition. Thus, the rules of the Seventh Mission of the International Aerial Robotics Competition have been studied.

The second part of the work has been to obtain a Simulation Platform adapted to our requirements. The UPM team had already developed a Simulation Platform for the IARC Seventh Mission. The simulator recreates the scenario of the competition virtually. Thanks to this recreation, the Mission Planner and other software components can be tested without all the hardware dependencies, neither recreating physically the entire scenario. The simulator simplifies drastically the development of those software components. Moreover, the simulator uses ROS, a middleware which provides a simple communication layer and allows creating easily a modular design. As ROS is also used

in the aerial vehicle, ROS modules are easy to migrate and the integration phase does not require special effort.

Although the simulator had great features for our project, another feature was required. For developing and evaluating new strategies the simulation platform needs to be executed thousands of times. The simulation time was too slow for the large number of executions. The simulation behavior was simulated as real-time simulation and it required ten minutes in the normal case, when the time is the condition to end the mission. To solve this, a faster mode has been developed for the Simulation Platform. The method used for this mode tries to minimize the amount of time when the processor is sleeping. In the real-time mode, the processor waits after a cycle is executed until the clock of next cycle advices it to advance. In the new mode, the waiting time is simulated. The simulated clock advances just when the computations of the cycle have finished. To achieve it, a synchronization method has to be used, changing slightly the architecture for communication between the different modules. The new mode developed achieves the same behavior but faster execution which depends on the computation power of the machine where it is executed. In our machine the execution speed-up reached has been 25.06 times faster than the original.

After providing a Simulation Platform which fits well with the requirements, Mission Planner strategies were developed. The aerial vehicle needs to take autonomously several decisions during the competition. The decisions proposed for the mission can be categorized in three different types. The first type is the searching strategies. A searching strategy decides where the UAV should go next in order to find robot candidates for guiding to the goal. The strategies to find robots include go to the center, go to a random point or move along a route of several points. In addition to these strategies, more complex strategies have been designed. For example, the planner decides to go to the last known position of a ground robot recently seen or combined strategies which tries to adapt to the changes of the behavior, among others. The second type of strategy is selection decision; the planner is in charge of deciding which robot is to be guided when there are several ones in its vision range. The strategies developed for this selection decision were related with the position of the ground robot. The last type of decision is addressed to discard ground robot from guidance. We have identified a couple of situations in which to discard a ground robot may improve the performance.

The last part of the work has been an evaluation of these strategies. The evaluation has used a large number of simulations executed with different strategies configurations. As each simulation is partially random, and then non-deterministic, the number of samples taken must be large enough to warranty enough reliability. The results of the executions were evaluated using some measurements of the performance of the entire execution. The main measurements used were the number of robots guided to the goal and the

mission completion rate. But also, other measurements have been used like the time to complete the mission or the number of robots disqualified to abandon the arena.

The simulations have shown interesting results. Firstly, the experiments allowed us to develop a planner strategy which shows high performance of the mission. The Diamond Route Searching Strategy in combination with the First Near Selection Strategy has achieved a performance of 7.9 robots guided in average, guiding at least seven in the 93% of simulations.

In addition to the selection of this strategy, the results show interesting facts related with the performance. The time of the mission is an important factor because it is limited. However, the experiments have shown that the importance of the time decreases through the mission elapses. At the last minutes of the mission losing time is not as important as at the beginning. Another significant knowledge acquired thanks to the experiments is that a performance improvement is achieved giving high priority to guide first those ground robots more near to the goal. Moreover, central part of the arena has been shown to be a highly significant area for searching robots. Finally, route strategies have shown best performance than the others.

During the work we have identified several points to continue working at. On the Simulator Platform, the main improvement in the execution time has already done. There are some possible optimizations like remove networking and ROS delays or try to parallelize computations. However, the computation would still need several seconds for each simulation. But another feature may be interesting. The ground robots act with partially random behavior, each five seconds the robots turns slightly randomly. It may be useful that those random behaviors would be loaded in order to reproduce specific situations. It does not mean that the simulation will become determinist, but just reduce the random factor impact. The ability to reproduce same situations can have many useful advantages, for example to allow testing one particular situation, or a representative set of them, with different strategies and analyze the differences.

In the case of development strategies, many new ideas can be used. For example, try large number of combinations changing the conditions to pass from one to another strategy. Following to this idea, use evolutionary algorithms is something that we would like to test. Also, thanks to the information obtained with our experiments we can try to refine those strategies which already work well. Finally, there is interesting to test more complex strategies, for instance a searching strategy which tries to predict the future position of ground robots.

Finally, on the evaluation stage, we can continue obtaining and analyzing more data. For instance, get data about the time elapsed to complete each action or get data about

the positions of the different agents. Additionally, a deeper analysis of the individual actions performed by the UAV may obtain new useful knowledge.

9. Bibliography

[1] R. Murphy, Introduction to AI Robotics, 2000.

[2] S. Bouabdallah, Design and Control of Quadrotors with Application to Autonomous Flying, 2007.

[3] T. B. Sherindan, Telerobotics, Automation, and Human Supervisory, MIT Press, 1992.

[4] R. Parasuraman, T. B. Sheridan, Fellow, IEEE and D. Wickens, "A Model for Types and Levels of Human Interaction".

[5] F. Kendoul, "A Survey of Advances in Guidance, Navigation and Control of Unmanned Rotorcraft Systems," 2012.

[6] M. R. Endsley, "Situation awareness in aviation systems," *Handbook of Aviation Human Factors, Publication of Lawrence Erlbaum Associates,* 1999.

[7] M. R. Endsley, Design and Evaluation for Situation Awareness Enhancement, 1988.

[8] IARC, "IARC Past Missions," [Online]. Available: http://www.aerialroboticscompetition.org/pastmissions.php. [Accessed February 2015].

[9] IARC, "Official Rules for the International Aerial Robotics Competition Mission 7 v11.0," 2014.

[10] A. M. Alcaraz, "Sistema de soporte al desarrollo de un planificador de misiones en un vehículo aéreo no tripulado," UPM Bachelor Thesis, 2014.

[11] J. Sanchez-Lopez, J. Pestana, J. Cullumeau, R. Suarez-Fernandez, P. Campoy and M. Molina, "Vision Based Aerial Robot solution for the Mission 7 of the International Aerial Robotics Competition," *International Conference on Unmanned Aircraft Systems ICUAS,* 2015.

[12] Open Source Robotics Foundation, "ROS - Robot Operating System," [Online]. Available: http://www.ros.org/. [Accessed February 2015].

[13] D. Coleman, I. Sucan, S. Chitta and N. Correll, "Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study," 2014.

[14] T. Foote, "tf: The Transform Library," 2013.

10. LIST OF ACRONYMS

ACL	Autonomous Control Level
AEF	Autonomy Enabling Functions
AFRL	Air Force Research Laboratory
ALFURS	Autonomy Levels For Unmanned Rotorcraft Systems
ALFUS	Autonomy Levels For Unmanned Systems
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
EC	Environmental Complexity
GNC	Guidance, Navigation and Control
GPS	Global Position Satellite
GUI	Graphical User Interface
HI	Human Independence
IARC	International Aerial Robotics Competition
MC	Mission Complexity
MIT	Massachusetts Institute of Technology
NIST	National Institute of Standards and Technology
OODA	Observe, Orient, Decide and Act
OOP	Object Oriented Programming
PR	Personal Robots
QNX	Quantum Software Systems
RAM	Random-Access Memory
ROS	Robot Operating System
RPC	Remote Procedure Call
RROS	RobotBASIC Robot Operating System
RUAS	Rotorcraft Unmanned Aerial System
STAIR	STanford AI Robot
UAV	Unmanned Aerial Vehicle
UPM	Universidad Politécnica de Madrid
URDF	Universal Robotic Description Format
XML	eXtensible Markup Language